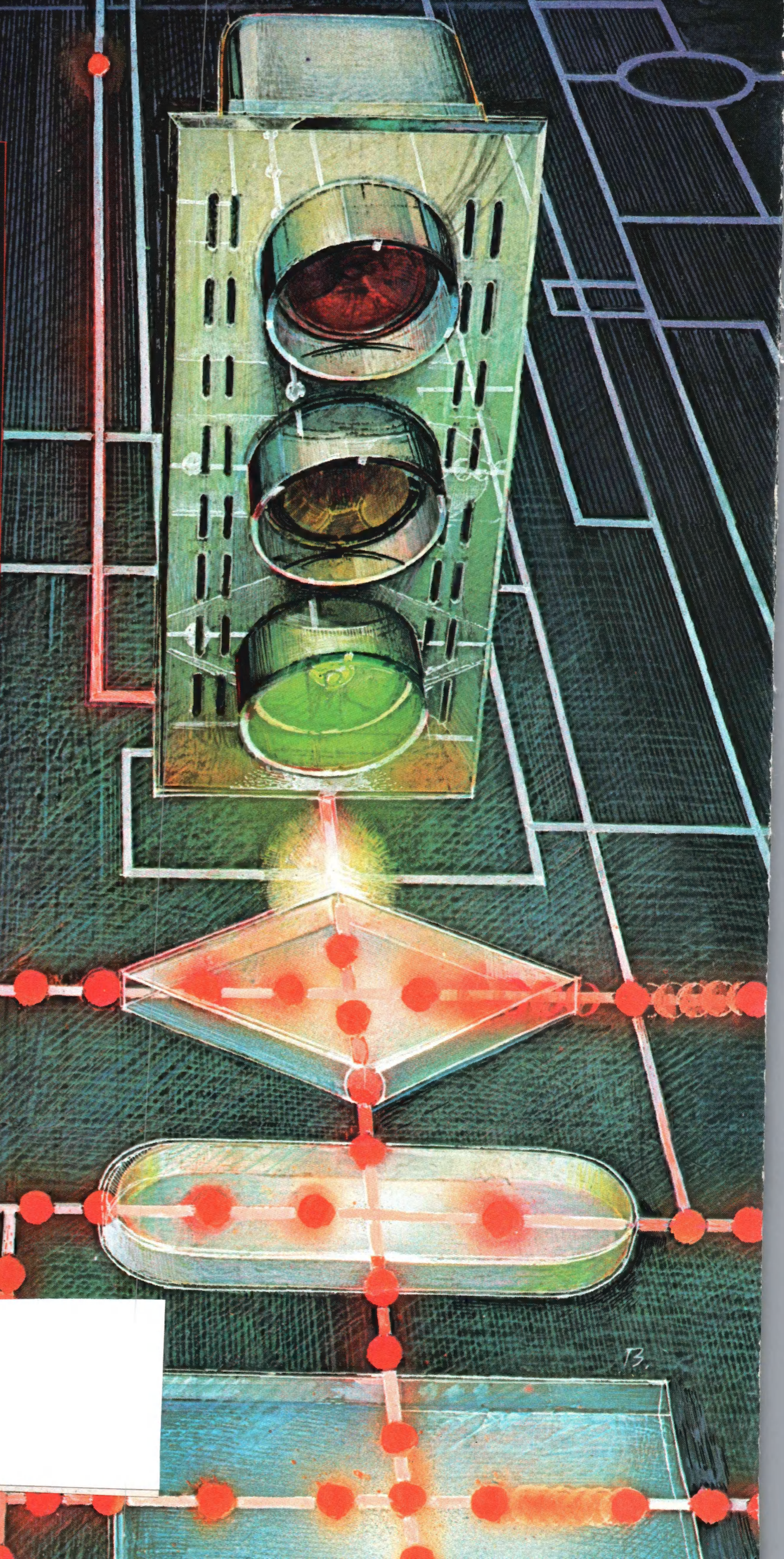
#128 JUNE 1987

2.95 (3.95 CANADA)

Dr. Dobb's Journal of Software Tools

FOR THE PROFESSIONAL PROGRAMMER



Handling Large Priority Queues

TSR Serial Driver

Analog-to-Digital Conversion

Unix Shell Scripts

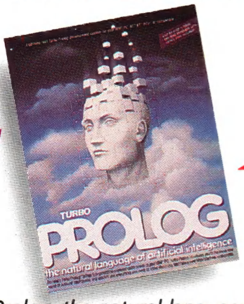
Languages:

Dynamic Overlays in Turbo Pascal
Handling Queues in C
AI Programming in SCOOPS
User-Defined Functions in BASIC

0 38351 16562 8

Turbo Prolog™

> “If you're at all interested in artificial intelligence, databases, expert systems, or new ways of thinking about programming, by all means plunk down your $100 and buy a copy of Turbo Prolog.
> **Bruce Webster, BYTE**”



Only $99.95!

Turbo Prolog, the natural language of Artificial Intelligence, is the most popular AI package in the world with more than 100,000 users. It's the 5th-generation computer programming language that brings supercomputer power to your IBM PC and compatibles. You can join the AI revolution with Turbo Prolog for only $99.95. Step-by-step tutorials, demo programs and source code included.

New! Turbo Prolog Toolbox

NEW

Our new Turbo Prolog Toolbox™ enhances Turbo Prolog—with more than 80 tools and over 8,000 lines of source code that can easily be incorporated into your programs. It includes about 40 example programs that show you how to use and incorporate your new tools.

New Turbo Prolog Toolbox features include:

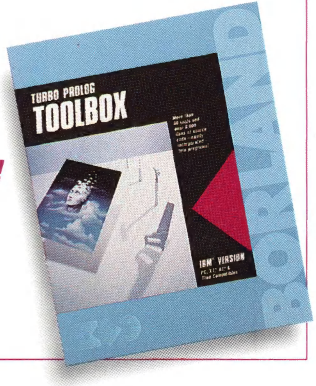
- ☑ Business graphic generation
- ☑ Complete communications package
- ☑ File transfers from Reflex, dBASE III, 1-2-3, Symphony
- ☑ A unique parser generator
- ☑ Sophisticated user-interface design tools

System requirements

Turbo Prolog: IBM PC, XT, AT or true compatibles. PC-DOS (MS-DOS) 2.0 or later. 384K. Turbo Prolog Toolbox requires Turbo Prolog 1.10 or higher. Dual-floppy disk drive or hard disk. 512K.

It's the complete developer's toolbox and a major addition to Turbo Prolog. You get a wide variety of menus—pull-down, pop-up, line, tree and box—so you can choose the one that suits your application best. You'll quickly and easily learn how to produce graphics; set up communications with remote devices; read information from Reflex,® dBASE III,® Lotus 1-2-3® and Symphony® files; generate parsers and design user interfaces. All of this for only $99.95.

Only $99.95!



MACINTOSH™ VERSION ALSO AVAILABLE

Turbo Pascal®

The power and high performance of Turbo Pascal is already in the hands of more than half-a-million people. The technically superior Turbo Pascal is the de facto worldwide standard and the clear leader.

The Turbo Pascal family includes:

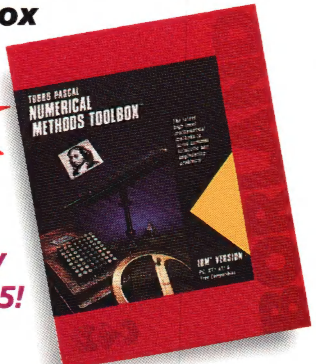
- Turbo Pascal® 3.0
- Turbo Tutor® 2.0
- Turbo Database Toolbox®
- Turbo Editor Toolbox®
- Turbo Graphix Toolbox®
- Turbo GameWorks®
- Turbo Pascal Numerical Methods Toolbox™

Turbo Pascal, the worldwide standard in high-speed compilers, and family.

> “The language deal of the century.
> **Jeff Duntemann, PC Magazine**”

New! Turbo Pascal Numerical Methods Toolbox

NEW



Only $99.95!

What our new Numerical Methods Toolbox will do for you now:

- ☑ Find solutions to equations
- ☑ Interpolations
- ☑ Calculus: numerical derivatives and integrals
- ☑ Differential equations
- ☑ Matrix operations: inversions, determinants and eigenvalues
- ☑ Least squares approximations
- ☑ Fourier transforms

As well as a free demo FFT program, you also get Least Squares Fit in 5 different forms:

1. Power
2. Exponential
3. Logarithm
4. 5-term Fourier
5. 5-term Polynomial

They're all ready to compile and run.

All this for only $99.95!

System requirements

IBM PC, XT, AT or true compatibles. PC-DOS (MS-DOS) 2.0 or later. Turbo Pascal 2.0 or later. Graphics module requires graphics monitor with IBM CGA, IBM EGA, or Hercules compatible adapter card, and requires Turbo Graphix Toolbox. 8087 or 80287 numeric co-processor not required, but recommended for optimal performance. 256K.

Turbo Pascal 3.0

Includes 8087 & BCD features for 16-bit MS-DOS and CP/M-86 systems. CP/M-80 version minimum memory: 48K; 8087 and BCD features not available. 128K.

BI-1103A

NEW **Turbo Basic**®

NOW SHIPPING!

Introducing Turbo Basic, the high-speed BASIC you'd expect from Borland!

It's the BASIC compiler you've been waiting for. And it's so fast that you'll never have to wait again.

Turbo Basic is a complete development environment; it includes a lightning-fast compiler, an interactive editor, and a trace debugging system.

Because Turbo Basic is compatible with BASICA, chances are that you already know how to use Turbo Basic.

With Turbo Basic your only speed is "Full Speed Ahead"!

You probably already know us for both Turbo Pascal® and Turbo Prolog.™ Well, we've done it again!

We created Turbo Basic, because BASIC doesn't have to be slow.

In fact, building fast compilers is a Borland specialty; both our Turbo Pascal and our Turbo Prolog outperform all their rivals by factors, and with Turbo Basic, we're proud to introduce the first high-speed BASIC compiler for the IBM® PC. If BASIC taught you how to walk, Turbo Basic will teach you how to run!

The Critics' Choice

"Borland has succeeded in stretching the language without weighing us down with unnecessary details . . . Turbo Basic is the answer to my wish for a simple yet blindingly fast recreational utility language . . . The one language you can't forget how to use, Turbo Basic is a computer language for the missus, the masters, the masses, and me.

Steve Gibson, InfoWorld

Borland's Turbo Basic has advantages over the Microsoft product, including support of the high-speed 8087 math chip.

John C. Dvorak"

Turbo Basic ends the basic confusion

There's now one standard: Turbo Basic.

It's fast, BASICA-compatible, and because Turbo Basic is a Borland product, the price is right, the quality is there, and the power is at your fingertips. You see, Turbo Basic's part of the fast-growing Borland family of programming languages—we call it the "Turbo Family." Hundreds of thousands of users are already using Borland's languages, so you can't go wrong. So join a whole new generation of smart IBM PC users—get your copy of Turbo Basic today. You get an easy-to-read 300+ page manual, two disks, and a free MicroCalc spreadsheet—and an instant start in the fast new world of Turbo Basic. All of this for only $99.95—Order your copy of Turbo Basic today!

Free spreadsheet included, complete with source code!

Yes, we've included MicroCalc, our sample spreadsheet, complete with source code, so that you can get started right away with a "real program." You can compile and run it "as is," or modify it.

A technical look at Turbo Basic

- ☑ Full recursion supported
- ☑ Standard IEEE floating-point format
- ☑ Floating-point support, with full 8087 (math co-processor) integration. Software emulation if no 8087 present
- ☑ Program size limited only by available memory (no 64K limitation)
- ☑ EGA and CGA support
- ☑ Access to local, static, and global variables
- ☑ Full integration of the compiler, editor, and executable program, with separate windows for editing, messages, tracing, and execution
- ☑ Compile, run-time, and I/O errors place you in the source code where error occurred
- ☑ New long integer (32-bit) data type
- ☑ Full 80-bit precision
- ☑ Pull-down menus
- ☑ Full window management

System requirements

IBM PC, XT, AT and true compatibles, PC-DOS (MS-DOS) 2.0 or later. One floppy drive, 256K.

Only $99.95!

BI-1103A

NEW **Turbo C**®

Turbo C: The fastest, most efficient and easy-to-use C compiler at any price

Compilation speed is more than 7000 lines a minute, which makes anything less than Turbo C an exercise in slow motion. Expect what only Borland delivers: Quality, Speed, Power and Price.

Turbo C: The C compiler for amateurs and professionals

If you're just beginning and you've "kinda wanted to learn C," now's your chance to do it the easy way. Like Turbo Pascal, Turbo C's got everything to get you going.

If you're already programming in C, switching to Turbo C will considerably increase your productivity and help make your programs both smaller and faster. Actually, writing in Turbo C is a highly productive and effective method—and we speak from experience. Eureka: The Solver and our new generation of software have been developed using Turbo C.

Turbo C: a complete interactive development environment

Free MicroCalc spreadsheet with source code

Like Turbo Pascal and Turbo Prolog, Turbo C comes with an interactive editor that will show you syntax errors right in your source code. Developing, debugging, and running a Turbo C program is a snap.

Turbo C: The C compiler everybody's been waiting for. Everybody but the competition

Borland's "Quality, Speed, Power and Price" commitment isn't idle corporate chatter. The $99.95 price tag on Turbo C isn't a "typo," it's real. So if you'd like to learn C in a hurry, pick up the phone. If you're already using C, switch to Turbo C and see the difference for yourself.

System requirements

IBM PC, XT, AT and true compatibles. PC-DOS (MS-DOS) 2.0 or later. One floppy drive. 320K.

Technical Specifications

- ☑ *Compiler: One-pass compiler generating linkable object modules and inline assembler. Included is Borland's high performance "Turbo Linker." The object module is compatible with the PC-DOS linker. Supports tiny, small, compact, medium, large, and huge memory model libraries. Can mix models with near and far pointers. Includes floating point emulator (utilizes 8087/80287 if installed).*
- ☑ *Interactive Editor: The system includes a powerful, interactive full-screen text editor. If the compiler detects an error, the editor automatically positions the cursor appropriately in the source code.*
- ☑ *Development Environment: A powerful "Make" is included so that managing Turbo C program development is highly efficient. Also includes pull-down menus and windows.*
- ☑ *Links with relocatable object modules created using Borland's Turbo Prolog into a single program.*
- ☑ *ANSI C compatible.*
- ☑ *Start-up routine source code included.*
- ☑ *Both command line and integrated environment versions included.*

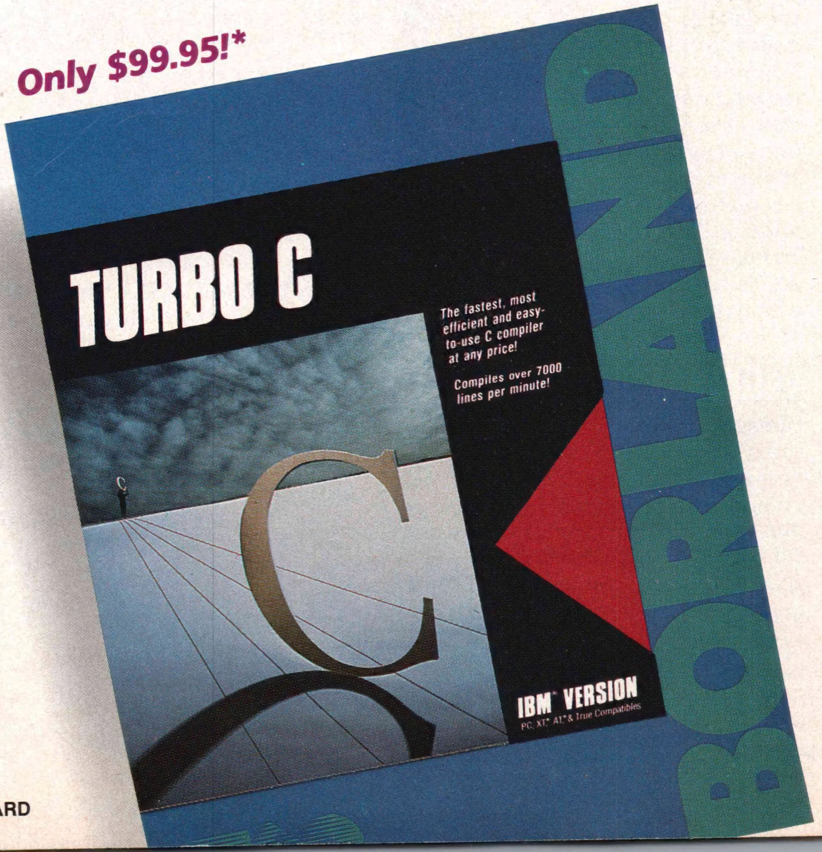
*Introductory price—good through July 1, 1987

Sieve benchmark (25 iterations)

	Turbo C	Microsoft® C	Lattice C
Compile time	**3.89**	16.37	13.90
Compile and link time	**9.94**	29.06	27.79
Execution time	**5.77**	9.51	13.79
Object code size	**274**	297	301
Price	**$99.95**	$450.00	$500.00

Benchmark run on a 6 Mhz IBM AT using Turbo C version 1.0 and the Turbo Linker version 1.0; Microsoft C version 4.0 and the MS overlay linker version 3.51; Lattice C version 3.1 and the MS object linker version 3.05.

Only $99.95!*



For the dealer nearest you, or to order by phone
call (800) 255-8008
CA (800) 742-1133
Canada (800) 237-1136

CIRCLE 161 ON READER SERVICE CARD

CONTENTS

VOLUME 12, ISSUE 6

ARTICLES

Efficient queue control

ALGORITHMS: An Efficient Algorithm for Large Priority Queues 16
by Robert Jay Brown
By deferring decisions about lower-level priorities, this algorithm is able to assign the top priority more efficiently.

COMMUNICATIONS: Two-Bit Analog-to-Digital Conversion 22
by John Musselman
How to use hardware interrupts and a demonstration of techniques common in real-time programming

Mathematical basis for a space-saving trick

ALGORITHMS: The XOR Chain 28
by David E. Cortesi
The Swapped-Out Intern returns with a technique that exploits a curious property of doubly linked lists.

TSR serial driver

COMMUNICATIONS: An Extended COM Port Driver 42
by Thomas A. Zimniewicz
Excom is a terminate-and-stay-resident program that gives the IBM PC interrupt-driven buffered input with flow-control selection and support for higher baud rates.

Handling large Turbo Pascal programs

LANGUAGES: Dynamic Memory Overlays for Turbo Pascal 50
by Steve McMahon
Steve shows how to get around the 64K limit on executable code imposed by Turbo Pascal without resorting to slow disk overlays.

Unix telecommunications software

COMMUNICATIONS: A Unix BBS Using Shell Scripts 54
by Jan L. Harrington
Jan shows just how she wrote a bulletin board system using Unix V Bourne shell scripts and XMODEM.

COLUMNS

Queue control in C

C CHEST 102
by Allen Holub
Allen delves into the subject of priority queues and, in Flotsam and Jetsam, discusses standard *#include* files.

For the programmer's bookshelf

16-BIT SOFTWARE TOOLBOX 112
by Ray Duncan
Ray discusses programming books. He also offers a poor man's MAKE utility and an MS-DOS programming tip.

Object-oriented programming

ARTIFICIAL INTELLIGENCE 116
by Ernest R. Tello
Ernie presents some examples of object-oriented programming using SCOOPS, an extension of PC Scheme, "the Turbo Pascal of the PC LISP family."

FORUM

Time to hock the Volvo

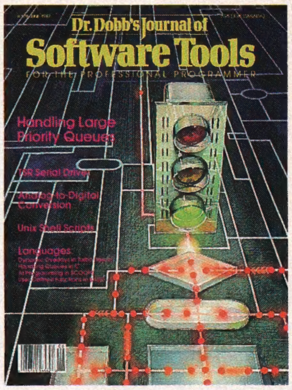
EDITORIAL 6
by Michael Swaine
RUNNING LIGHT 8
by Levi Thomas
ARCHIVES 8
LETTERS 10
by you

What's right with high-level languages?

VIEWPOINT 14
by Brian R. Anderson
SWAINE'S FLAMES 136
by Michael Swaine

PROGRAMMER'S SERVICES

THE STATE OF BASIC: 128
A comparison of user-defined functions in four BASICs
OF INTEREST: 132
Products for programmers
ADVERTISER INDEX: 113
Where to find those ads



About the Cover

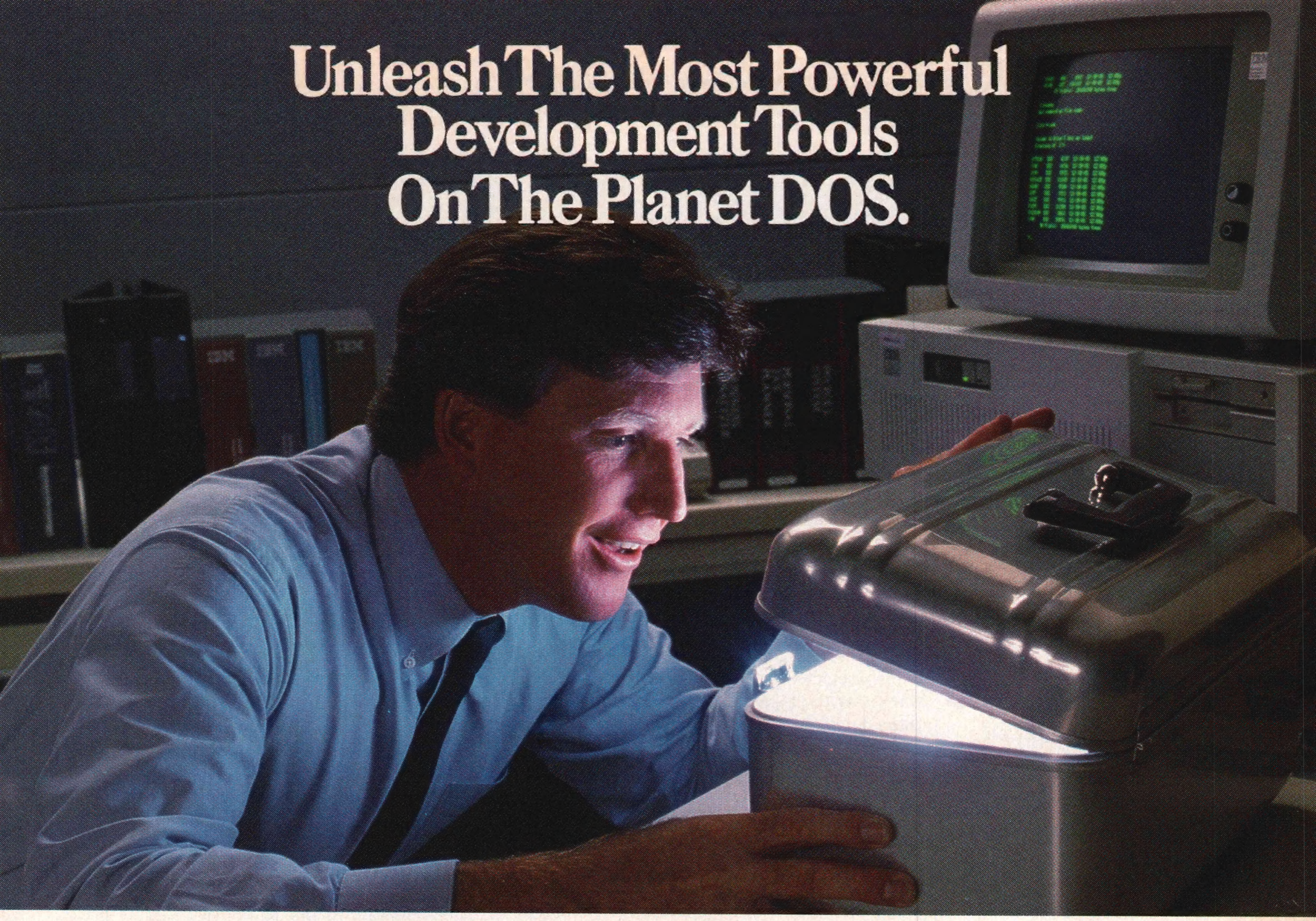
Handling the traffic flow of multitasking requires an efficient queueing algorithm such as the one that has just given the green light to a task in this month's cover illustration.

This Issue

If this issue looks packed, it is. Three articles deal with communications techniques. Robert Jay Brown and Allen Holub discuss algorithms for priority queues. Dave Cortesi, a longtime *DDJ* columnist, contributes a rare bit on doubly linked lists. Steve McMahon gives aid and comfort to serious Turbo Pascal programmers; Ernie Tello and Allen Holub give tips to new PC Scheme and C programmers, respectively; and Forth, assembly-language, Modula-2, and BASIC programmers will all find something of interest in this issue.

Next Issue

So you really did it? You sold the Volvo and bought a 386 machine and now you're going to develop software for it? Only you don't want to wait for Microsoft to deliver OS/2 sometime next year? We hope you saved $2.95 for our July issue, in which we will discuss development tools for the 386 that are available today.



UNIFY DBMS/DOS. The UNIX World Leader Brings A New Dimension To DOS Application Development.

What happens as the DOS world expands? As a new generation of hardware takes over? As networking becomes more important? The potential is enormous. But until now, the tools to achieve it have been limited.

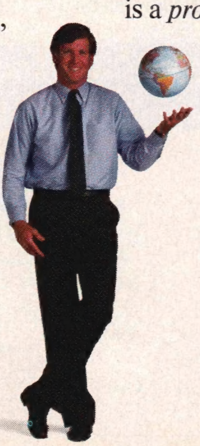
Now a leader from another world unleashes that potential: UNIFY® DBMS. The leading relational DBMS in the UNIX™ world. And now, the most advanced set of application development tools in the DOS world.

With UNIFY DBMS, DOS developers have new power to build more sophisticated applications than ever before possible.

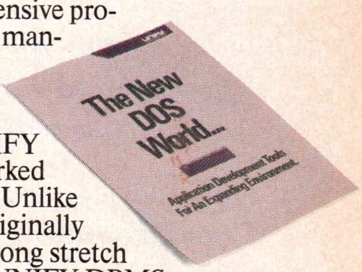
The power to write high performance "C" programs that will access the data base, using Unify's Direct Host Language Interface.

The power of an industry standard query language—SQL.

The power of unmatched speed in production applications. Only UNIFY DBMS is specifically engineered for transaction throughput. With unique performance features like PathFinder™ Architecture multiple access methods, for the fastest possible data base access.



The power of comprehensive program development and screen management tools. Plus a state-of-the-art fourth generation report-writer.



What's more, with UNIFY DBMS, the potential of networked applications becomes a reality. Unlike DBMS systems which were originally single-user (and which have a long stretch to accommodate more users), UNIFY DBMS is a *proven* multi-user system.

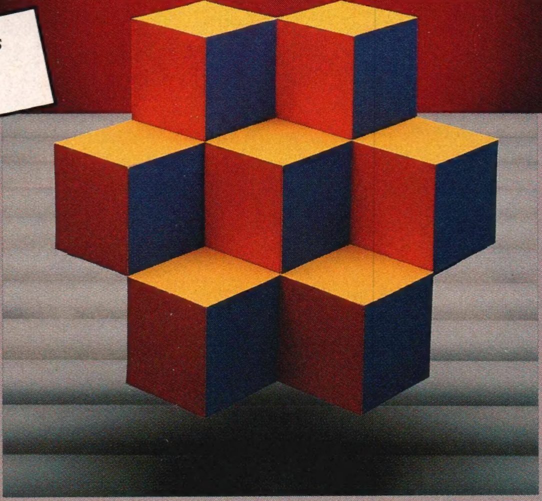
And because UNIFY DBMS/DOS is the best of two worlds, it offers you the most powerful benefit of all: DBMS applications that can grow as your needs grow. From single user DOS. To networked DOS. To multi-user UNIX. All without changing your applications.

Call the Unify Information Hotline for our free booklet: The New DOS World. (503) 635-7777

UNIFY
CORPORATION
4000 Kruse Way Place
Lake Oswego, OR 97034

WHY LOGITECH MODULA-2 IS MORE POWERFUL THAN PASCAL OR C.

"A clear winner... The integrated editor is a joy to use."
BYTE Magazine,
Jan. '87



APPRENTICE PACKAGE
$99

WIZARDS' PACKAGE
$199

- **Separate Compilation w/inter-module typechecking**
- **Native Code Generation**
- **Large Memory Model Support**
- **Most Powerful Runtime Debugger**
- **Comprehensive Module Library**
- **Maintainability**
- **Translator from Turbo and ANSI Pascal**

WIN A FREE TRIP TO Switzerland

HOMELAND OF MODULA-2

Return your Modula-2 Registration Card or a reasonable facsimile,* postmarked between March 1, 1987 and May 31, 1987 to be included in a once-only drawing!

Grand Prize: One week excursion for 2 in Zurich, Switzerland including a guided tour of ETH, the University where Modula-2 was created by Niklaus Wirth. European customers may substitute a trip to Silicon Valley, California.

Second and Third Prizes: LOGITECH C7 Mouse or LOGITECH Bus Mouse with Paint & Draw software—a $219 value, absolutely free!

*Write to Logitech, Inc. for a registration card facsimile.

NEW!

APPRENTICE PACKAGE $99

Everything you need to begin producing reliable maintainable Modula-2 code. Includes the Compiler with 8087 support, integrated Editor, Linker, and BCD Module. We're also including FREE our Turbo Pascal to Modula-2 Translator!

NEW!

WIZARDS' PACKAGE $199

This package contains our Plus Compiler—for professional programmers or for those who just want the best. The Plus Compiler with Integrated Editor requires 512K and takes advantage of the larger memory to increase compilation speed by 50%. Our Turbo Pascal to Modula-2 Translator is also included at no extra charge.

NEW!

MAGIC TOOLKIT $99

We've put our most powerful development tools into one amazing Toolkit for use with either the Apprentice or Wizards' packages. Highlighted by our Runtime Debugger, the finest debugging tool available anywhere, the Toolkit also includes our Post Mortem Debugger, Disassembler, Cross Reference utility and Version which keeps track of different versions of one program. Our MAKE Utility figures out module dependencies and automatically selects those affected by code changes to minimize recompilation and relinking. We also provide source code of our major library modules for you to customize—or just play with.

WINDOW PACKAGE $49

Now you can build true windowing into your Modula-2 code. Features virtual screens, color support, overlapping windows and a variety of borders.

ROM PACKAGE AND CROSS RUN TIME DEBUGGER $299

For those who want to produce rommable code. You can even debug code running in ROM from your PC.

Turbo Pascal is a registered trademark of Borland International.

Call for information about our VAX/VMS version, Site License, University Discounts, Dealer & Distributor pricing.

To place an order call toll-free:

800-231-7717

In California:

800-552-8885

YES! I want the spellbinding power of LOGITECH Modula-2!

- ☐ Apprentice Package **$99**
- ☐ Wizards' Package **$199**
- ☐ Magic Toolkit **$99**
- ☐ Window Package **$49**
- ☐ ROM Pkg/Cross RTD **$299**

Add $6.50 for shipping and handling. Calif. residents add applicable sales tax. Prices valid in U.S. only.

Total Enclosed $________

☐ VISA ☐ MasterCard ☐ Check Enclosed

Card Number ____________ Expiration Date ________

Signature ____________

Name ____________

Address ____________

City ____________ State ________

Zip ____________ Phone ________

LOGITECH

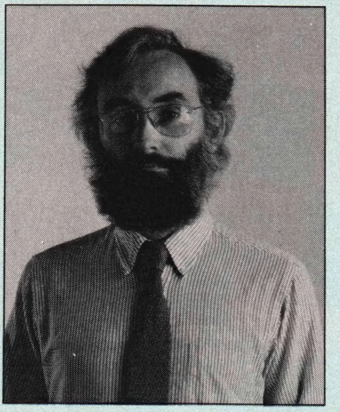
LOGITECH, Inc.
805 Veterans Blvd. Redwood City, CA 94063
Tel: 415-365-9852

In Europe:
LOGITECH SA, Switzerland
Tel: 41-21-879656 • Telex 458 217 Tech Ch

In Italy:
Tel: 39-2-215-5622

CIRCLE 257 ON READER SERVICE CARD

EDITORIAL



The summer of 1987. Midyear. Time to take stock; time to assess what we've seen of 1987 and mull over what it all means. What can we say? That it was the spring of hype, the summer of virtuality? I think it's a watershed summer, a good time to be a software developer.

Microsoft, IBM, and Apple all claimed that we were entering a new generation of personal computing. Both Apple and IBM announced significant new machines. Both Apple and IBM committed to fast, smart new buses and more open architectures, which will create new challenges and opportunities for hardware and software developers.

Microsoft revealed plans for getting parts of its OS/2 for 80286 and 80386 processors into developers' hands this year. Ever since Gary Kildall put an operating system on an 8080, the only arguably revolutionary advance in personal computer operating systems has been the Macintosh system software. It might be a stretch to suggest that OS/2 could be the next, but when it really arrives it should prove fertile ground for software development.

Both Unix and C took steps toward standardization in the first half of this year, and the crowded C compiler market ripened toward some sort of mitosis. Both Unix and C presented new tools and new environments for development. Version management and CASE became more real for personal computer-based development.

If you're reading this in Tokyo, you already know that several Japanese computer companies have taken a big step toward a viable Japanese-user-oriented personal computer industry by standardizing on the TRON real-time operating nucleus. TRON should have a much greater impact on Japanese computer users than the Fifth Generation Project, and it's also going to have a significant impact on software developers by providing an entirely new, coherent, Japanese-script-based programming platform.

These new environments and new tools speak of new opportunities for the software developer. But they speak in the ambient din, and it's easy to miss their message.

If you're currently evaluating the software development opportunities presented by these recent advances, could I make some modest suggestions?

Don't listen too closely to the marketers. Marketing experts are magicians; when it's difficult to create value, marketers can create need. They can convince people that there is something terribly wrong with them that only your product can cure. But it's so easy to create real value with a piece of software that I'm just not sure that we need the digital deodorant. If the product is good, marketing it becomes a simpler matter.

And I wouldn't listen too closely to users, heretical as that statement may be. Listening to users doesn't generally contribute a lot to creating something new. User feedback is useful when you're tweaking existing products. Users may not know anything about software, but they know what they don't like.

Ultimately, I argue, the real breakthroughs come from listening to the technology. That should be particularly true at a technological watershed like this summer.

Hock the Volvo, buy that 68020 or 80386 machine, and start playing around.

Michael Swaine

Michael Swaine
editor-in-chief

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Dr. Dobb's Journal of Software Tools (USPS 307690) is published monthly by M&T Publishing Inc., 501 Galveston Dr., Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points.

Article Submissions: Send manuscripts and disk (with article and listings) to the Editor.

***DDJ* on CompuServe:** Type GO DDJ

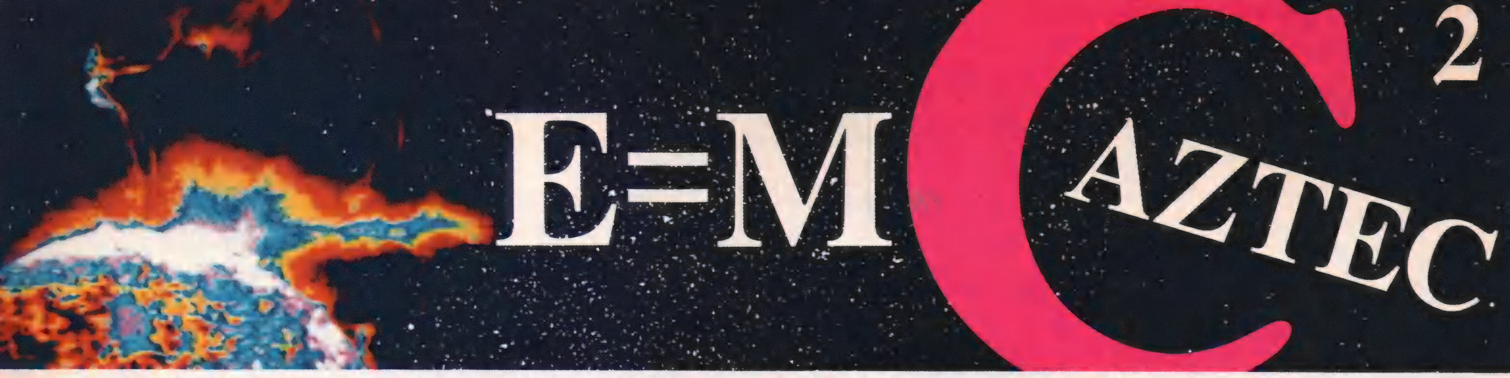
Address Correction Requested: Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, P.O. Box 27809, San Diego, CA 92128. **ISSN 0888-3076**

Customer Service: For subscription problems call: outside CA (800) 321-3333; in CA (619) 485-9623 or 566-6947. For book/software order problems call (415) 366-3600.

Subscriptions: $29.97 per 1 year; $56.97 for 2 years. Canada and Mexico add $27 per year airmail or $10 per year surface. All other countries add $27 per year airmail. Foreign subscriptions must be prepaid in U.S. funds drawn on a U.S. bank. For foreign subscriptions, TELEX: 752-351.

Foreign Newsstand Distributor: Worldwide Media Service Inc., 386 Park Ave. South, New York, NY 10016; (212) 686-1520 TELEX: 620430 (WUI).

Dr. Dobb's Journal of Software Tools is published by M&T Publishing Inc. under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a nonprofit corporation.



Our thanks to NASA for supplying this computer enhanced ultraviolet photo taken by Skylab IV of a solar prominence reaching out 350,000 miles above the sun's surface

Genius Begins With A Great Idea ...

But The Idea Is Just The Beginning

What follows is the time consuming task of giving form and function to the idea.

That's why we concentrate on building into our software development systems functions and features that help you develop your software ideas in less time and with less effort.

We've started 1987 by releasing new versions of our MS-DOS, Macintosh, Amiga, ROM, and Apple // C development systems. Each system is packed with new features, impressive performance, and a little bit more genius.

Aztec C86 4.1
New PC/MS-DOS • CP/M-86 • ROM

Superior performance, a powerful new array of features and utilities, and pricing that is unmatched make the new Aztec C86 the first choice of serious software developers.

Aztec C86-p Professional System $199
• optimized C with near, far, huge, small, and large memory + Inline assembler + Inline 8087/80287 + ANSI support + Fast Float (32 bit) + optimization options • Manx Aztec 8086/80x86 macro assembler • Aztec overlay linker (large/small model) • source level debugger • object librarian • 3.x file sharing & locking • comprehensive libraries of UNIX, DOS, Screen, Graphics, and special run time routines.

Aztec C86-d Developer System $299
• includes all of Aztec C86-p • Unix utilities make, diff, grep • vi editor • 6 + memory models • Profiler.

Aztec C86-c Commercial System. $499
• includes all of Aztec C86-d • Source for library routines • ROM Support • CP/M-86 support • One year of updates.

Aztec C86 Third Party Software

A large array of support software is available for Aztec C86. Call or write for information. The following is a list of the most requested products: Essential Graphics • C Essentials • C Utility Library • Greenleaf Com. • Greenleaf General • Halo • Panel • PC-lint • PforCe • Pre-C • Windows for C • Windows for Data C terp • db Vista • Phact • Plink86Plus • C-tree.

CP/M • TRS-80 • 8080/Z80 ROM

C compiler, 8080/Z80 assembler, linker, librarian, UNIX libraries, and specialized utilities.

Aztec C II-c (CP/M-80 & ROM). $349
Aztec CII-d (CP/M-80) $199
Aztec C80 (TRS-80 3&4) $199

Aztec C68k/Am 3.4
New Amiga Release

Amiga user groups across the USA voted Aztec C68k/Am release 3.3 the best Software Development System for the Amiga. Release 3.4 is more impressive.

Aztec C68k/Am-p Professional $199
A price/feature/performance miracle. System includes: optimized C • 68000/680x0 assembler • 68881 support • overlay linker • UNIX and Amiga libraries • examples.

Aztec C68k/Am-d Developer $299
The best of Manx, Amiga, and UNIX. System includes: all of Aztec C68k/Am-p • the Unix utilities make, diff, grep and vi.

Aztec C68k/Am-c Commercial. $499
Aztec C68k/Am-d plus source for the libraries and one year of updates.

Aztec C68k/Mac 3.4
New Macintosh Release

For code quality, reliability, and solid professional features, Aztec C for the Macintosh is unbeatable. This new release includes features and functions not found in any other Macintosh C development system.

Aztec C68k/Mac-p Professional $199
• optimized C • 68000/680x0 assembler • 68881 support • overlay linker • UNIX and Macintosh libraries • examples.

Aztec C68k/Mac-d Developer. $299
The best of Manx, Macintosh, and UNIX. System includes: all of Aztec C68k/Am-p • the Unix utilities make, diff, grep • vi editor.

Aztec C68k/Mac-c Commercial $499
Aztec C68k/Am-d plus source for the libraries and one year of updates.

Aztec C65
New ProDOS Release

Aztec C65 is the only commercial quality C compiler for the Apple II. Aztec C65 includes C compiler, 6502/65C02 assembler, linker, library utility, UNIX libraries, special purpose libraries, shell development environment, and more. An impressive system.

Aztec C65-c Commercial $299
• runs under ProDOS • code for ProDOS or DOS 3.3

Aztec C65-d Developer. $199
• runs under DOS 3.3 • code for DOS 3.3

Aztec ROM Systems
6502/65C02 • 8080/Z80 • 8086/80x86 • 680x0

An IBM or Macintosh is not only a less expensive way to develop ROM code, it's better. Targets include the 6502/65C02, 8080/Z80, 8086/80x86, and 680x0.

Aztec C has an excellent reputation for producing compact high performance code. Our systems for under $1,000 outperform systems priced at over $10,000.

Initial Host Plus Target $750
Additional Targets $500
ROM Support Package $500

Vax, Sun, PDP-11 ROM HOSTS

Call for information on Vax, PDP-11, Sun and other host environments.

C' Prime
PC/MS-DOS • Macintosh
Apple II • TRS-80 • CP/M

These C development systems are unbeatable for the price. They are earlier versions of Aztec C that originally sold for as much as $500. Each system includes C compiler, assembler, linker, librarian, UNIX routines, and more. Special discounts are available for use as course material.

C' Prime $75

Aztec Cross Development Systems

Most Aztec C systems are available as cross development systems. Hosts include: PC/MS-DOS, Macintosh, CP/M, Vax, PDP-11, Sun, and others. Call for information and pricing.

How To Become An Aztec C User

To become a user call 800-221-0440. From NJ or international locations call 201-542-2121. Telex: 4995812 or FAX: 201-542-8386. C.O.D., VISA, MasterCard, American Express, wire (domestic and international), and terms are available. One and two day delivery available for all domestic and most international destinations.

Aztec C is available directly from Manx and from technically oriented computer and software stores. Aztec Systems bought directly from Manx have a 30 day satisfaction guarantee.

Most systems are upgradable by paying the difference in price plus $10. Site licenses, OEM, educational, and multiple copy discounts are available.

MANX

To order or for more information call today.

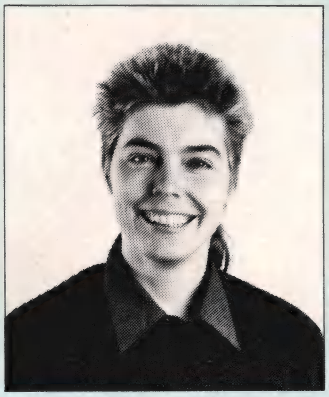
1-800-221-0440

In NJ or international call (201) 542-2121 • TELEX: 4995812

CIRCLE 108 ON READER SERVICE CARD

Manx Software Systems
1 Industrial Way, Eatontown, NJ 07724

RUNNING LIGHT



Greetings, programmers. I'm Levi Thomas, assistant editor and host of DDJFORUM, our SIG on CompuServe. If my writing style seems conversational, it's because I'm accustomed to the SIG, on which my readers can talk back.

It's an interesting phenomenon, the on-line life. When I first began working for *DDJ* as electronic editor I felt like a DJ on the graveyard shift. Posting messages on the message board of the SIG, I got the eerie feeling that there was really nobody out there, that just maybe I was talking to myself. Luckily the FORUM members are a verbal bunch, and even the lurkers can be coaxed into responding if I ask the right questions or start a topic that gets their dander up.

Now I want to talk to you hard-copy readers. I wanna pull on yer coats about something. I hope you'll talk back to me, either on line or via the U.S. Snail.

<hopping onto soapbox>

Microcomputers have grown up.

We all know the story of the microcomputer's Wonder Bread Years. Born with features only a mother could love (front panel toggle switches, 2K of RAM, no permanent storage, and so on), they have matured into smart, fast machines.

The microcomputer community has also grown up. What started a few years ago as a hobby has become a business. The community has become an industry. For many of you, your fascination with computers has turned into a shot at The Dream Job: A chance to make a living at something you love to do. A chance to make work into play.

However, like many new adults, this industry seems determined to disown all of its childlike qualities as it seeks a grown-up identity. It's been giving up many of the values that fueled its growth. The Suits have moved in, and things are tightening up.

Examples: It's becoming next to impossible to get a job on pure ability; college degrees are the new prerequisite. The sense of community we once got from the computer magazines is disappearing. I see the term *hacker* changing—not evolving into the essence of what it originally meant but devolving into a derogatory term, a term feared by this new, "grown-up" industry. (Last year I was involved with the Hacker's Conference. Several computer businesses were nice enough to help fund the affair, but some of them asked that their names not be disclosed. They were nervous about having their company names associated with the word *hacker*.)

<returning soapbox to closet>

DDJ is still a reader-responsive magazine. If you would like to write for our October (Forth), November (graphics), or December (operating systems) issues, send us an outline. You can address your article proposals to our new editor, Tyler Sperry (you'll meet him on this page next month). If you just can't stand the thought of using the U.S. Sail, you can e-mail me on CompuServe (76703,4060), Usenet (well!levi), or arpanet (well!levi@lll-crg.arpa). We look forward to hearing from you.

Levi Thomas

Levi Thomas
assistant editor

ARCHIVES

Inaugural Columns

"Welcome to the first edition of Programming Pastimes and Pleasures. Together we will explore the prosaic and fantastic possibilities of programming. Sometimes I will pose a problem to solve; sometimes I will discuss a programming technique or idea you may not have heard of; at other times I will tell you about a game you can play with your computer. At times I will be serious, and at other times I will be off-the-wall. Along the way, you will learn things you didn't know before, and you will find wonderful new ways to waste time with your computer. Most of all, I hope you will have fun."—*Charles Wetherell*, DDJ, *January 1979*.

"Doctor Dobb's Clinic is a new venture. It is a place for the display of techniques and discoveries. We want to cover any sort of method or trick you've found in your exploration of (for example, but not limited to): CP/M or OASIS or FLEX or APPLE DOS or TRSDOS or NEWDOS or RSTS or UNIX or UCSD PASCAL or....

"During our rounds at the clinic we may examine a compiler or an interpreter, revealing its bugs or showing how it can be made to perform better. We'd like to tell of unobvious uses for standard utilities. We want to uncover errors in published documentation, to warn people away from pitfalls, and to show off those 'eureka!' moments that make systems work rewarding."—*Dave Cortesi (Resident Intern)*, DDJ, *May 1981*.

"Just after watching Doc Dobb's nationally televised speech from the NCC in Texas, I was surprised to receive a collect call from the Old Man himself. 'The lack of public domain software for the 16-bit microcomputers is appalling!' remarked our Fearless Leader. 'No one ants to shell out that kind of money just so they can stare at the operating system's sign-on message. I want you to institute a regular column in *DDJ* that will address the needs of 16-bit system users and promote the discussion and interchange of software.' Luckily, after laying down that sweeping and rather alarming mandate, the good Doctor had to hang up to go out on a house call (someone had choked on an Apple) and I was left to my own devices."—*Ray Duncan*, DDJ, *September 1982*.

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia
Running Light Without Overbyte

LETTERS

What's Right with High-Level Languages?

Dear *DDJ*,
Mike Suman's criticisms of Modula-2 in the February 1987 Viewpoint call for some comments.

First, although many Modula-2 implementations are now available, based on Wirth's *Programming in Modula-2* (either the second or third edition), both the language specifications and the preferred standard libraries are still being fine-tuned by such bodies as M2WG, the Modula-2 Working Group of the British Standards Institution. As I understand the situation, draft proposals from BSI now await reaction and further input via the ISO. *MODUS Quarterly* (published by the Modula-2 Users' Association) has been covering this debate and provides a "democratic" forum for suggestions, including Wirth's own reactions. Mike should throw in his pen'orth before the stonecutters start chiseling. Contact George Symons, MODUS, P.O. Box 51778, Palo Alto, CA 94303; (415) 322-0547.

Second, I must defend Brian Anderson's use of the identifiers *FIRST* and *LAST* in X68000. Although in the final, published version these turn out to be nonce constants, during the development stage (and later when the MC68030 is incorporated!) who knows what values they might assume or where else they might occur?

Finally, you must really sympathize with Niklaus. You want to keep a language "simple," "portable," and "small-core," yet hardly is the ink dry before committees emerge wanting to add their pet piece of FORTRAN or Ada. Where do you draw the line? Certainly array and record constants would be useful (as in Turbo but not standard Pascal) and may find their way into the canon. Modula-2 does have the useful, non-Pascal *Module Body* construct (a piece of code invoked just once when the module is first executed).

Mike's preferred tabular layout for setting up Brian Anderson's *Table68K* will be possible under my own forthcoming language called Modula 1-2-3.

Stan Kelly-Bootle
25 Parkwood Ave.
Mill Valley, CA 94063

Stan Kelly-Bootle is the author of The Modula-2 Primer. *—eds.*

Dear *DDJ*,
In his February 1987 Viewpoint, Mike Suman raises a couple of objections that he suggests show something wrong with Modula-2 and other high-level languages. I don't think his objections are cogent, and I don't think they have anything to do with high-level languages.

His first complaint concerns the use of constants to replace Arabic integers. This is not peculiar to high-level languages but is a feature of most assembly languages, too, so whatever is wrong, it can't be a problem of Modula-2 or of high-level languages in general.

His particular complaint concerns the use of *FIRST* and *LAST* to replace 1 and 118, respectively; in the program on which he is commenting, they appear in the declaration:

```
Table68K : ARRAY[FIRST . . LAST] OF
                         TableRecord;
```

This does look odd, but the usual use of constants in this situation would omit the constant *FIRST* and replace the constant *LAST* by something more informative, such as *NumberOfOpcodes*, so the declaration would look like this:

```
Table68K : ARRAY[1 . . NumberOfOp-
              codes] OF TableRecord;
```

A scarlet letter?

I can't see why he would object to that. He does say, "no one can maintain that it is really easier, or safer, to change values in an early definition than it is to change them in the only place in which they are used later, when it is obvious what the effects of the change are going to be." But six months and five revisions later, how will you know that the number of opcodes is used only once in the program, and how easy will it be to find that one place?

His larger complaint is that Modula-2 contains no compact and readable way to initialize a complex table. But then, what language does?

There are really two points mixed up here. One is that Modula-2 provides no way to initialize variables other than by assignment statements; there are no facilities for de-

Architecture of Wendin's Concurrent I/O System

Wendin's Operating System Toolbox makes it possible to write a concurrent driver for any I/O device.

Wendin's Operating System Toolbox kernel is composed of three major components: the scheduler, the memory manager, and the I/O subsystem. This month we will show how the I/O subsystem works, at the QIO (Queued Input/Output) device driver level.

The I/O system supports concurrent and waited I/O operations from multiple processes simultaneously. Its architecture was derived from the VAX/VMS QIO system, which was developed by Digital Equipment Corporation with these ideas in mind. The whole QIO mechanism takes advantage of many of the kernel's capabilities, including ASTs (Asynchronous System Traps), and event flags (which are used to signal I/O completion).

Before a program can perform I/O operations on a device, it must assign a channel to the device. A channel is simply a number, similar to an MS-DOS file handle. The operating system uses an assigned channel number as an index into an array of channel control blocks (CCBs) to get information about the channel and the corresponding device.

After assigning a channel with the SYS_ASSIGN system service, the user program can make calls to SYS_QIO or SYS_QIOW (for concurrent or waited I/O) through the INT FF software interrupt. The kernel calls the routine EXEQIO, which validates the caller's parameters and I/O privileges. If the call parameters are valid and the channel number corresponds to an assigned CCB, EXEQIO will deliver a KERNEL mode AST to the routine ASTQIO in the same process context. This allows the driver to run on its own stack, but in the context of the calling process. ASTQIO receives all information about the I/O operation in an IRP (I/O Request Packet), created by EXEQIO from the user program's parameters.

```
void ast_qio (packet)
  struct irp *packet;
{
  struct ccb *c;
  struct ucb *u;
  void (*driver)(struct irp *);

  c = &io_chantbl [packet->chan];
  u = &io_unittbl [c->unit];
  driver = u->driver;
  (*driver)(packet);
} /* ast_qio */
```

Listing 1. How ASTQIO dispatches the device driver given a channel number.

ASTQIO uses the channel number to find the corresponding CCB, and obtain a pointer to a data structure describing the device assigned to, in the format of a unit control block (UCB). The unit number is used as an index into an array of UCBs, much as the channel number is used as an index into an array of CCBs. The UCB contains information about a specific device, including what general type of device is being referenced (for example, a terminal, mailbox, or disk), and a pointer to the device driver code. This double indirection (the channel obtains a CCB, which in turn references a UCB, which finally references the device driver) allows several channels to be assigned to the same device at the same time, and for any number of UCBs to share the same device driver code (for example, the disk driver defines a UCB for each disk drive, but there is only one device driver for all of them).

Once ASTQIO has obtained the UCB for the requested device, it calls the device driver directly. The device driver is responsible for determining what to do based on the I/O function given in the IRP. The device driver will typically call the system BIOS, or interface directly to the hardware, to perform an I/O operation.

Finally, the kernel must signal the user process that the I/O is done. The SYS_QIO services allow two methods for this: an event flag can be set, or an AST can be executed when the operation finishes. The kernel routine STOPIO examines the IRP associated with an operation, and determines which event flag to set, and whether to queue an AST to the user process. Finally, the IRP is removed from the system I/O queues, and returned to the memory pool. Control is then passed back up the chain and returns to the user program.

A device driver can be written to talk to any device, based on standard models provided by the Operating System Toolbox kernel. Examples include mailboxes (functionally equivalent to UNIX pipes), terminals, disk drives, optical disks, or even expanded or extended memory. Basically, the minimum I/O functions that the driver needs to support are the read and write block functions. For some devices (like disks) there will be a distinction between virtual, logical, and physical I/O. For others (like mailboxes), all of these functions will perform the same operation. You can decide how you want a device driver to work, and it's easy to integrate a driver into the toolbox kernel.

Next month, we'll continue with the internals of the toolbox. If you'd like to learn more about operating system design and architecture, pick up a copy of Operating System Toolbox from Wendin and follow along.

Operating System Toolbox: $99
Wendin, Inc.
P.O. Box 3888
Spokane, WA 99220
(509) 624-8088

(continued from page 10)

fining initial values at the point of declaration as in some other languages (although the compiler I use provides this as an extension). Unless the compiler is very clever, this may mean some unnecessary work at execution time, but this does not seem like a major problem and certainly not one inherent in high-level languages.

The second point is that it is tedious to initialize a large table (in the example Suman discusses, it consists of 118 records of 4 fields each) with assignment statements. But there are other ways. For instance, I like to write a table in a readable form in an ASCII disk file and read it at run time to initialize the table. It's hard to beat this for readability, the table can be edited without recompiling the program, and the code to read the table is usually not difficult to write. There is a problem with reading in enumerated types from an ASCII file: because it can be a nuisance to translate the ASCII version of the identifiers for the values, you can compromise by using Arabic numbers for their ordinal values instead. This can be endured because the table file can easily contain a key for the values.

Suman offers an assembly-language version of the table in which sets are initialized by 16-digit binary numbers. I suppose it's a matter of taste, but I don't find 118 entries like that very readable, and I'm sure I'd make plenty of errors typing them into my program. And what would he do if he had a table with real numbers in it?

I suppose it would be nice if we could write our tables in readable tabular form in our code and have the compiler do the work, but there are lots of tasks we might like our compilers to perform, and we can't build them all in. I would guess that initializing complex tables is one of those tasks that is just as well left out of general-purpose languages.

John G. Bennett
301 Roslyn St.
Rochester, NY 14619-1813

Dear *DDJ*,
Though the Viewpoint by Mike Suman is interesting and raises some valid points, I find it to be misleading in three specific ways.

First, the point about the constants *FIRST* and *LAST* being used only once is not quite right. Consider the index variable *i*. The initialization and manipulation of *i* should always be in the range *FIRST . . LAST*, and further, it should exactly cover this range. The program is poorly written in that the index should have been initialized to *FIRST*, not 1 explicitly. Next, the value of the index at the end of the initialization should have been checked to see that it was equal to *LAST*, so that you know all the elements of the table have been initialized. Far from pointing out a place in which Modula-2 is too strict, this points out a place in which the programmer just didn't use the language properly.

Second, the fact that Modula-2 doesn't allow the specification of constant arrays is not an indication that Modula-2's type checking is too strict or that its model of programming is inadequate. It means that Modula-2 doesn't have this feature. It is clear that allowing constant aggregates to be uttered in the language is an independent issue from whether the language is inconveniently strict or not. Strictness and features provided are two completely different issues.

Third, asking if imagined (or even real) problems with the Modula-2 language mean there is "something wrong with the direction in which we are being led" is to miss the point altogether. The point of "higher-level" languages that have a strict type structure is not to prevent programmers from full expression or to make it harder to write legal programs. The point is to help programmers by making it easier for them to tell when they have written a meaningless or illegal program. Here we see that a programmer misused the constant definition facility of the language and that the language lacks the ability to utter constant-valued aggregates. But neither of these faults is a fault of strict type checking, nor of modularity, nor of the high-level nature of the language.

So, if we are being led in the direction of modularity and static type safety, Mike hasn't even begun to present evidence that the direction is wrong. He has shown that small languages often lack features that are awkward to do without, and he has shown that people often forget that index variables ought to be initialized and manipulated using the same bounds that were used to declare the array they index into. These are indeed things to be wary of, and they show that Modula-2 is neither omnipowerful nor errorproof. But this is no reason to throw away the baby of type checking and modularity with the bathwater of a possibly too small language and human error.

Wayne Throop
86 Fearrington
Pittsboro, NC 27312

Happy Ducks

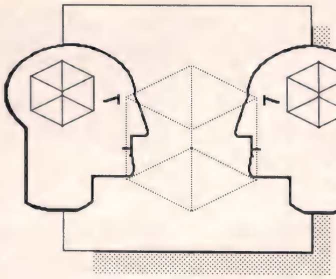
Dear *DDJ*,
Thank you for the very funny February cover revealing the true nature of us WordStar-imprinted programmers. But no thanks for the text editors article. For a *DDJ* feature article, it was amazingly uninformative—simply an excuse for a two-barrel discharge against us happy ducks.

Especially unfair was the statement that WordStar-style editors "usually use 'weird' file formats that can't be read by any other editor without some sort of conversion." WordStar itself works in straight ASCII when in its program-editing (nondocument) mode. But that is indeed not the most usual WordStar-style editor. The MUWSSE is, of course, the Turbo Pascal editor, which many waterfowl use with other compilers as well. And that is nothing but straight ASCII. So, just what were you talking about?

Another gross misstatement was the wish list. Every programmer I know, even strictly dry land, will tell you that wish number 1 is speed, 2 is speed, and 3 is speed. Other wishes appear only after pausing for breath. No mention of that in your list. Of course, again, the MUWSSE—the Turbo editor—is the fastest thing under ten fingers. And for the price of just about any dedicated programming editor, you can get Turbo Pascal and SuperKey, and then you have the

(continued on page 122)

VIEWPOINT



What's Right with High-Level Languages?

by Brian R. Anderson

In "What's Wrong with High-Level Languages" (Viewpoint, February 1987), Mike Suman brings up some interesting points about the limitations inherent in any computer language but fails to mention any of the strengths of Modula-2 or of modern high-level languages in general.

As a college instructor who has spent several years developing and teaching a course in assembly-language programming, I can well appreciate the advantages of low-level programming. Assembly language has some shortcomings that can be much more serious than the limitations of high-level languages, however.

Assembly language is much harder to learn than are most high-level languages, and you must substantially relearn it if you move from one computer to another. I grant that the second assembly language is much easier to learn than the first. It would, however, take a considerable effort to adapt from the 68000 to the 8086, for instance, especially when you consider the peculiarities of Microsoft's MASM.

Assembly language is hard to write and even harder to debug (compared to Modula-2), partially because often the only control structures available are the conditional and unconditional jump and the call to subroutine. To test several conditions for a loop, you must write a cascade of comparisons and jumps. To produce a simple repetition, you must split the control instructions between the outside of the loop, the top of the loop, and the bottom of the loop. Consider Examples 1 and 2, below, in which some elements of a character array are cleared using 68000 assembly language and Modula-2, respectively. In both examples, the range of items to be cleared has been calculated and left in variables named *TOP* and *BOT*. In the assembly-language example, the loop constraints are developed in five different lines of code, whereas in the high-level language example, all loop constraints are developed on a single line.

Modern high-level languages allow programmers to express data in terms that naturally match a wide range of typical problems. Modula-2 handles the mathematical concepts of real and integral numeric types and sets, the organizational concepts of records and files, and the universal concepts of arrays and characters. In assembly language, the only data type is the computer word. Programmers must impose a structure and then provide algorithms to perform even the simplest task. Consider how much easier it is (for exam-

(continued on page 124)

```
                          **********************************************
                          *   FOR loop in 68000
                          *
                          *   First Array Element to Process: TOP
                          *   Last Array Element to Process: BOT
                          *   The array: DATA
                          *
                          *   Used to Index into array: DO
                          *   Used as Pointer to array: AO
                          *
                          **********************************************
                          *
000000   00               DATA     DS     500           ;array of bytes
0001F4   0000             TOP      DC     0             ;first to process
0001F6   0000             BOT      DC     0             ;last to process
                          *
                          *
                                                        ;other data
                                                        ;and code
                          *
                          *
                          *  Assumes TOP & BOT previously calculated
                          *
0001FB   41F900000000              LEA    DATA,AO       ;set pointer
0001FE   3039000001F4              MOVE   TOP,DO        ;init index
000204   B079000001F6     LOOP     CMP    BOT,DO        ;check bounds
00020A   6206                      BHI.S  LABEL         ;end loop
00020C   42300000                  CLR.B  O(AO,DO)      ;loop body
000210   60F2                      BRA.S  LOOP          ;repeat
                          LABEL
000212                             END
```

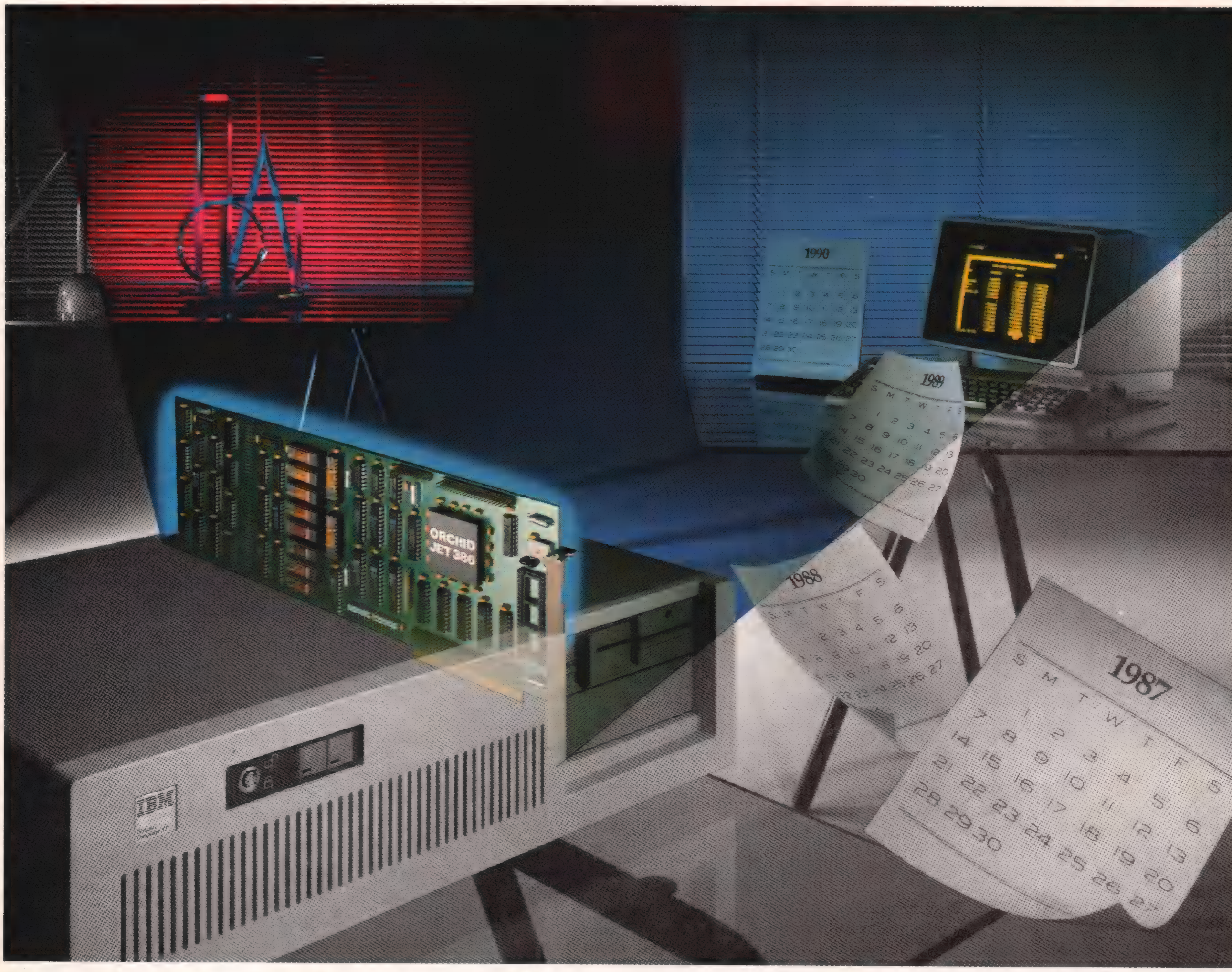
Example 1: *Clearing elements of a character array in 68000 assembly language*

```
MODULE TRIAL;

   VAR
      DATA : ARRAY [1..500] OF CHAR;
      TOP, BOT, i : CARDINAL;
   BEGIN
      (* Assumes TOP & BOT have been previously calculated *)

      FOR i := TOP TO BOT DO
         DATA[i] := OC;
      END;

   END TRIAL.
```

Example 2: *Modula-2 version of the code in Example 1*

Brian R. Anderson, 5105 Lorraine Ave., Burnaby, B.C. V5G 2S3 Canada. Brian is an instructor in the Electronics Dept. at the Vancouver Vocational Institute.

ORCHID'S JET 386™: POWER FOR THE FUTURE NOW



Jet 386 is the Ultimate Accelerator Upgrade for Your AT

Announcing an end to obsolescence. Orchid Technology's Jet 386™ accelerator card extends the life of your computer investment into the 1990s—it puts power in your AT that you won't outgrow.

Three Times Faster than an AT

It's up to three times faster than an AT depending on the application, and speed is just one benefit. Unequalled compatibility and provisions for upcoming 386 software mean your Jet 386 will handle whatever the future has in store: CAD, spreadsheets, networking…

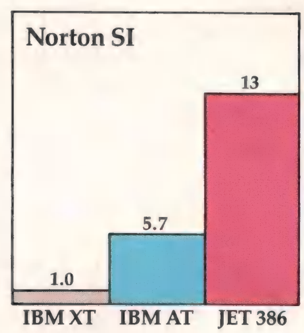
Easy Upgrade

Easy to use, there's nothing new to learn and no new programs to buy. At 25% of the cost of buying a new 386 PC, it's easy on your pocketbook, too.

From the People Who Started It All

Orchid combined 80386 power with the technology perfected for the XT in the TinyTurbo and PCturbo 286e. Like these critically acclaimed accelerators, Jet 386 is built for lasting value.

Call Orchid to find out how you can experience the future today. And ask how Orchid can modernize your whole office with turbos, graphics, networking, and multifunction products.



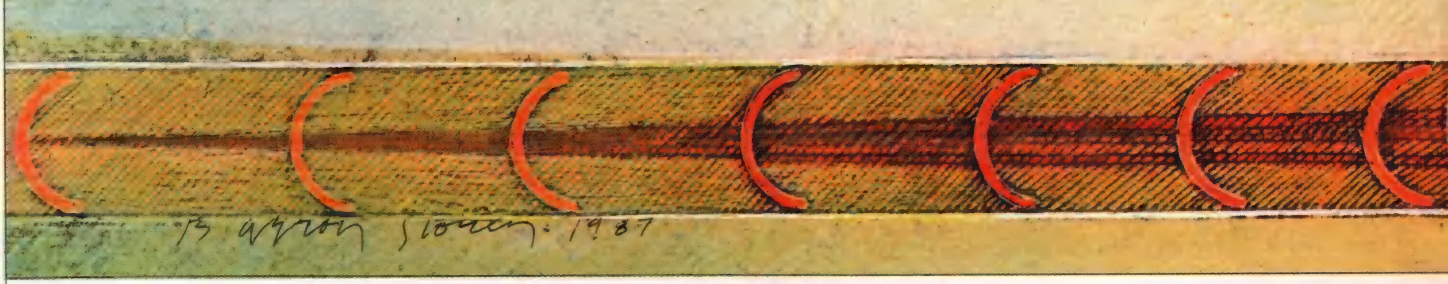
ORCHID
Innovative Add-Ons

45365 Northport Loop West
Fremont, CA 94538
415/490-8586 Tlx: 709289

CIRCLE 130 ON READER SERVICE CARD

An Efficient Algorithm for Large Priority Queues

by Robert Jay Brown



In many real-time programs, it is necessary to share resources that are in short supply. To help manage these situations, a priority system is often established. When several contenders are competing for the same resource, the one with the highest priority gets the resource. When the resource again becomes available, the next-priority contender gets it. A simple first-in/first-out queue can be thought of as a priority queue in which the time a contender is enqueued is the priority and lower numbers have higher priority.

In practice, the resource being waited on may be a physical device, such as a printer; a logical device, such as a file or a record of a file; or the CPU itself. Alternatively, a timer scheduler may be implemented by using the desired dispatch time as the priority and having the dequeueing operation wait until the time of day equals the dispatch time at the head of the queue.

The central concept is that inserting and removing from the queue can be viewed as merging two separate queues.

Basic Queue Operations

A priority queueing scheme in a real-time system must be able to perform the following operations on the elements, or nodes, of the queue: determine the highest-priority node, add a new node, and remove a node. Removing the highest-priority node is a special case of the more general operation of removing a node anywhere in the queue. This is called preempting.

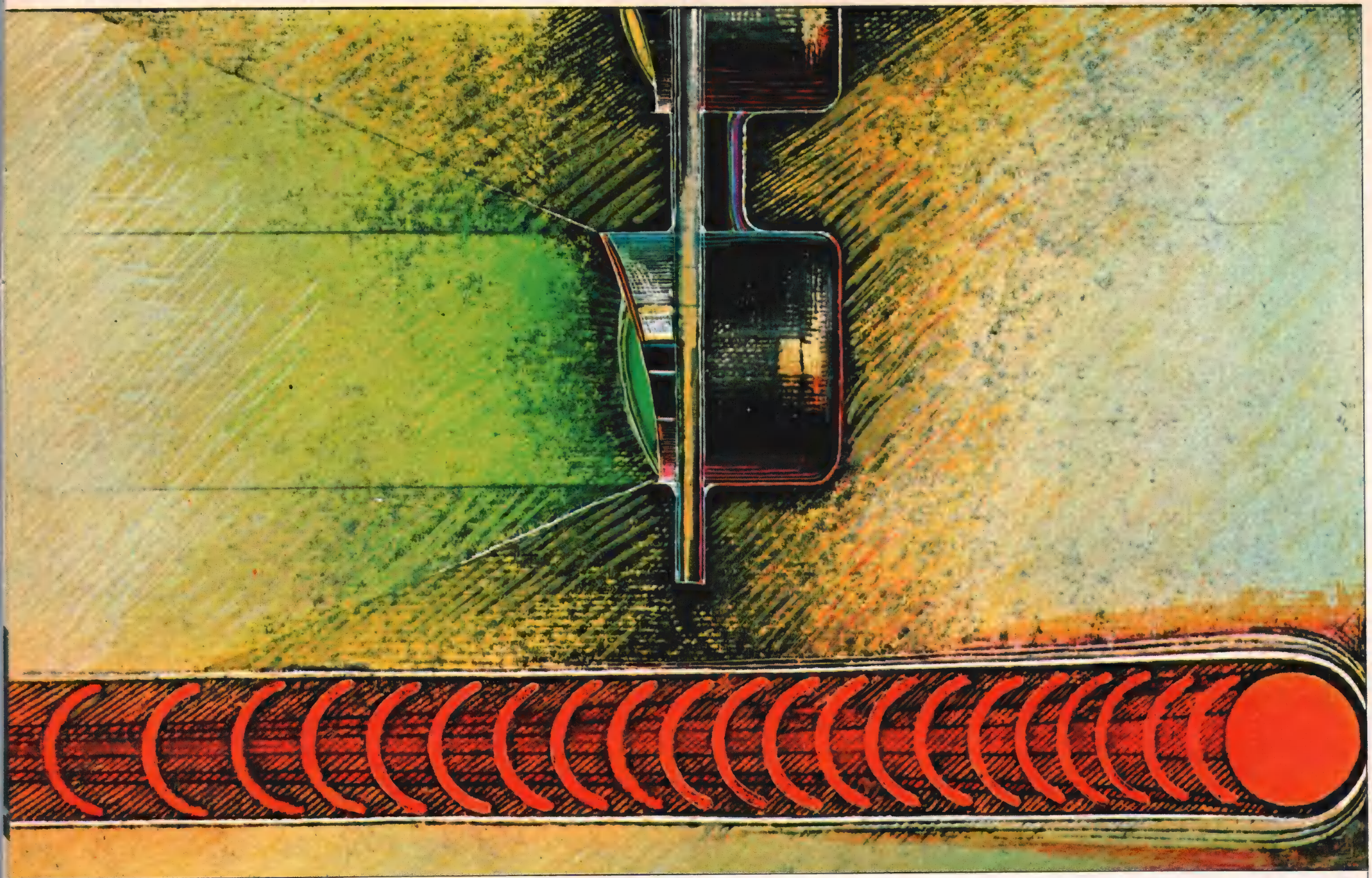
The most simple implementation for priority queues results in the time to perform at least one of the above operations being directly proportional to the size of the queue. For extremely large queues, this becomes unworkable. The problem is similar to sorting, and sorting can be done in a time proportional to the logarithm of the number of elements, so you should be able to do as well for a priority queue.

In fact, you can do a bit better than this. Although the priority queue algorithm I have implemented (see Example 1, page 18) performs in logarithmic time, it does not keep the queue completely sorted. It defers determining the second-priority element until after the top-priority element is removed. This does not change the logarithmic component, but it makes each iteration go faster.

A Binary Tree

The representation for the queue is a binary tree with *left*, *rite*, and *father* connections. Each element also con-

Robert Jay Brown, 301 W. High St., P.O. Box 833, Warsaw, KY 41095. Robert is a graduate student at Florida Atlantic University. He is a consultant involved in designing electronic surveillance intercept and cryptography systems.



tains a sort *key*, which can be interpreted as the priority. In addition, a *dist* cell is used to indicate the minimum distance from that node to a leaf, which is a connection to no element, or a terminator. The *father* connection is not used to maintain the ordering of the queue but is used to allow rapid removal of any node in the queue (see the example, lines 28–37). (Note: my structure declaring words [lines 3–26], together with examples of their use, are available on the East Coast Forth Board as the file STRUC.ARC, and so I will not describe them further here.)

The *word* and *data* cells (lines 36–37) are used to contain dispatching information. *Word* is the address of a Forth word to *perform* when the node is dispatched, and *data* is a word of data, typically a pointer, that is pushed on the stack before *perform*ing the *word*.

The following priority queueing algorithm was first described by Clark Allen Crane in 1971. My implementation is an extension of a revised version of Crane's algorithm that is described by Donald Knuth. You can refer to Knuth's *The Art of Computer Programming: Volume 3, Sorting and Searching* (Reading, Mass.: Addison-Wesley, 1973) for a complete description of the algorithm and an analysis of its performance.

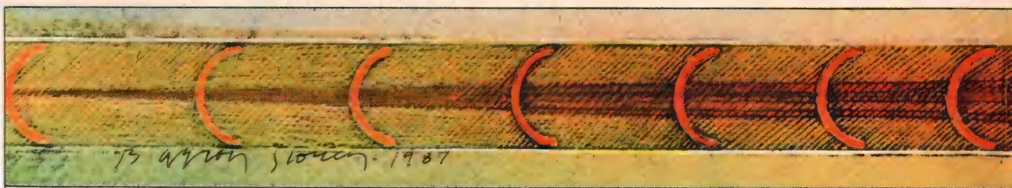
Merging Queues

The central concept behind Crane's algorithm is that the operations of inserting and removing an element from the queue can be viewed as merging two separate queues. Crane's algorithm keeps the highest-priority node at the root of the tree, and the subtrees follow the same pattern. Thus, to dispatch the head element of the queue (lines 103–106), the left and right subtrees are pruned from the root and merged back together, leaving the root out of the result. To insert an element on the queue (lines 98–101), that element is viewed as a little priority queue, in its own right, of one element. This queue is merged with the original queue, and the element is thereby inserted into the original queue. To remove a node from somewhere in the middle of the queue—that is, to preempt it (lines 108–111), the node's *left* and *rite* subtrees are cut off and merged, forming an intermediate result. Next, the preempted node is removed by using its *father* pointer to cut it off from the element that points to it. Finally, the original queue, less the preempted node and its two subtrees, is merged with the intermediate tree described above.

The implementation in the example has been tested on a 10-MHz 80286 with one wait state on memory access and 3.2 percent DRAM refresh interference running under LMI PC/FORTH+ 3.1. For 10,000 iterations of an *unque* on a randomly selected element of the queue, followed by an *enque* using a randomly chosen priority, it produced the results shown in Table 1, page 18.

Availability

The complete source code, as a standard Forth screen file, including the benchmark test, is available on CompuServe (GODDJ), and on the East Coast Forth Board ([703] 442-8695). Also, all the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr.*



QUEUEING ALGORITHM
(continued from page 17)

Dobb's Journal, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 2.**

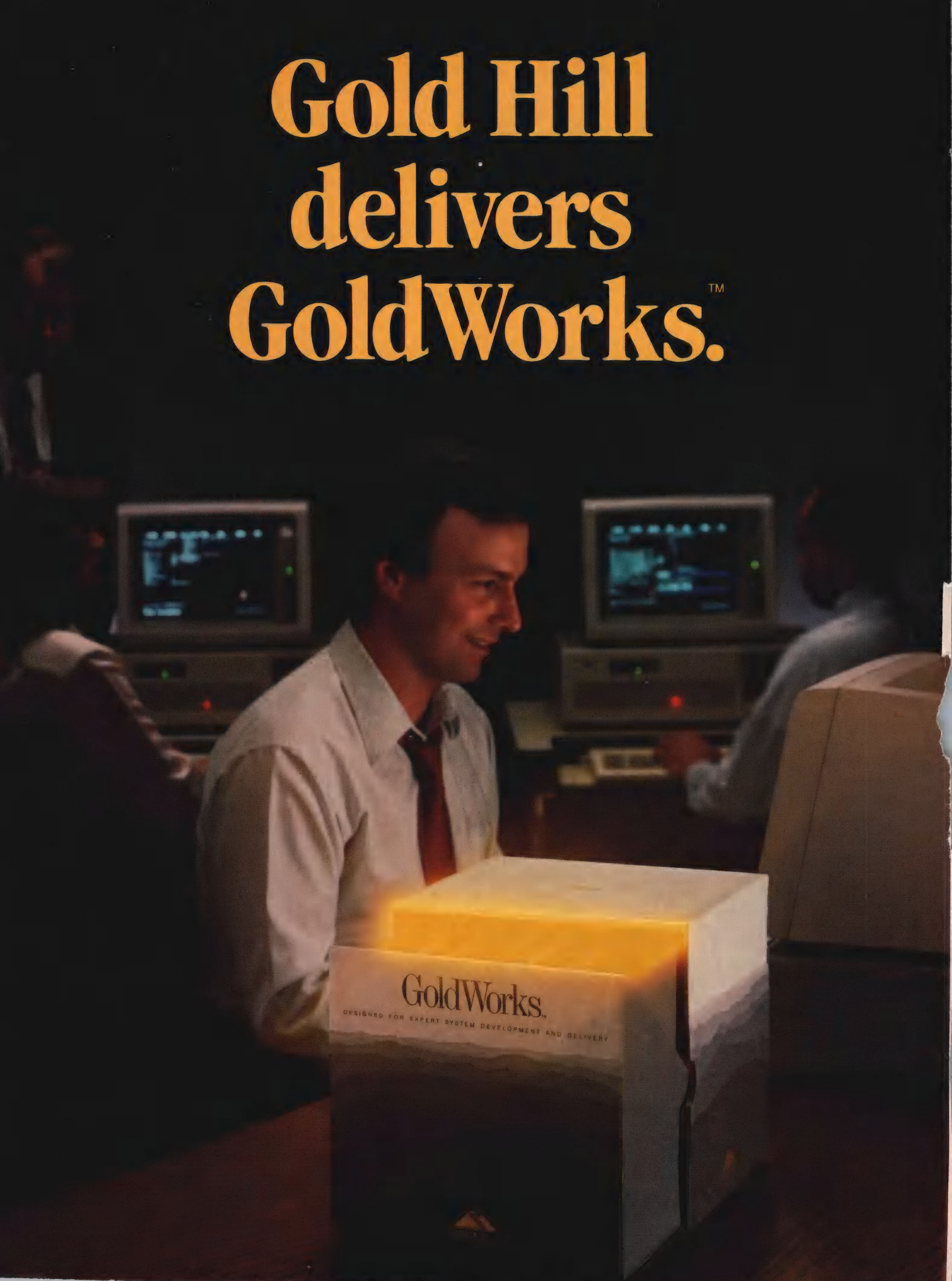
#nodes	msecs
10	7.4
20	9.3
50	11.9
100	13.8
200	15.5
500	17.9
1000	19.4
2000	20.7
5000	21.9
10000	23.1

Table 1: *Benchmark test results*

```
( Timer queue implementation                          rjb 16:06 07/2186 )

( -------------- structure declaring words ---------------- )

: <struc> CREATE , DOES> @ + ; ( 2nd generation defining word )

( 'struc' is used to create the defining word for a structure )
: struc CREATE ,     ( org struc <struc> ; creates a structure )
        DOES>    ( size <struc> <element> ; creates an element )
        DUP @ DUP >R ROT + SWAP !   ( update location counter )
        R> <struc> ;                  ( make word for element )

( 'array' is used to create an array of structures )
: array CREATE ( #slots slot-size array <array> ; makes array )
        DUP , * ALLOT       ( save slot-size and allocate array )
        DOES>           ( subscript <array> -- &<array>element )
        DUP @ ROT * + WSIZE + ;   ( return pointer to the slot )

( 'org' is useful for doing the 4th version of a C 'union' )
: org    ( n org <strucname> ; re-initializes location counter )
        BL WORD FIND NOT ABORT" Undefined! " >BODY ! ;

( 'sizeof' is used for declaring structures of structures )
: sizeof ( sizeof <strucname> -- n ; gets the size of a struc )
        BL WORD FIND NOT ABORT" Undefined " >BODY @
        STATE @ IF [COMPILE] LITERAL THEN ; IMMEDIATE

( ---------------- timer queue entry ---------------- )

    0   struc   tq                     ( a node in the timer queue )
        4       tq      key       ( the sort key: dispatch time )
        4       tq      dist         ( distance to nearest leaf )
        4       tq      left         ( pointer to left sub-tree )
        4       tq      rite        ( pointer to right sub-tree )
        4       tq      father      ( pointer to father of node )
        4       tq      word       ( word to execute at dispatch )
        4       tq      data         ( pointer to data for word )

( ---------------- timer queue implementation --------------- )

( left! & rite! set the left and rite subtrees, respectively
  of a node, called the father.  The father pointer of the son
  node is updated to point to the pointer, left or rite, that
  points to the son, so that it may be cleared on an unque. )

: left! DUP 0<> IF OVER left OVER father !     ( Father Son -- )
```

Example 1: *Timer queue implementation*

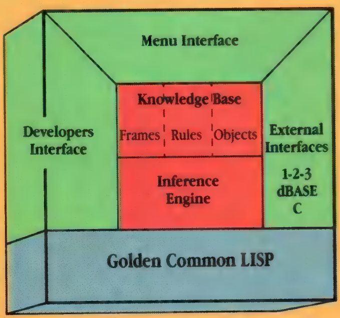
```
      THEN SWAP left ! ;

: rite! DUP 0<> IF OVER rite OVER father !     ( Father Son -- )
      THEN SWAP rite ! ;

: go-rt DUP rite @ >R DUP ROT rite! R> ;     ( go dn rite side )

: dist@ DUP IF dist @ THEN ;          ( P -- P=nil ? 0 : dist @ )

( This is step M2: from Knuth p 619, sol'n to prob. 32, p 159 )
: list-merge          ( P Q R -- P Q R D ; merge priques P & Q, )
    BEGIN  ( R is Roving pointer, D is Distance to near leaf )
      OVER 0= IF                                    ( Q = nil ? )
          2 PICK dist@ EXIT THEN  ( yes, D = P->dist; done! )
      2 PICK 0= IF                                  ( P = nil ? )
          ROT DROP OVER SWAP                        ( yes, P = Q )
          2 PICK dist@ EXIT THEN  ( yes, D = P->dist; done! )
      2 PICK key @   2 PICK key @       ( P->key < Q->key ? )
      < IF ROT go-rt ROT ROT ELSE        ( yes, P moves right )
           SWAP go-rt SWAP   THEN        ( no,  Q moves right )
   AGAIN ;        ( loop until one of the trees is eliminated )

( This is steps M3: and M4: from Knuth p 619 )
: fix-dist ( P Q R D -- P ; fixes up distance to nearest leaf )
  BEGIN
     OVER 0= IF 3DROP EXIT THEN       ( R = nil ?  yes, done! )
     ROT DROP OVER rite @ ROT ROT                ( Q = R->rite )
     OVER left @ dist@ OVER < IF       ( R->left->dist < D ? )
        DROP DUP left @ dist@ 1+      ( D = R->left->dist + 1 )
        OVER DUP left @ rite!            ( R->rite = R->left )
        OVER 4 PICK left!    ELSE                ( R->left = P )
        1+ OVER 4 PICK rite!  THEN        ( D++; R->rite = P )
     DUP 2 PICK dist !                           ( R->dist = D )
     >R ROT DROP SWAP DUP R>                    ( P = R; R = Q )
  AGAIN ;                       ( spin down the right sub-tree )

( tq-merge is Knuth's Algorithm M from p 619 )
: tq-merge 0 list-merge fix-dist ;  ( P Q -- P ; merge 2 tq's )

( the timer queue root pointer )             VARIABLE TQ 0 TQ !

( TQ! updates the father pointer in the first node, & sets TQ )
: TQ!                     ( tq -- ; sets TQ and father of tq )
       DUP TQ !                       ( set the timer queue head )
       DUP 0= IF DROP EXIT THEN             ( empty? yes, done! )
       father TQ SWAP ! ;         ( no, set father of top node )

: ?waiting DUP father @ @ = ;                       ( tq -- flag )

( ------------ timer queue package entry points ------------- )

: tq-enque                           ( &tq -- ; enques to timer )
      0 OVER left ! 0 OVER rite !   ( nullify both sub-trees )
      1 OVER dist !                  ( distance to a leaf is 1 )
      TQ @ tq-merge TQ! ;     ( merge new node with old queue )

: tq-deque                         ( -- &tq ; deques from timer )
      TQ @ DUP 0= IF EXIT THEN   ( returns nil on empty queue )
      DUP                                ( save head for answer )
      DUP left @ SWAP rite @ tq-merg TQ! ;    ( merge remains )

: tq-unque                    &tq -- &tq ; removes from timer )
      DUP ?waiting NOT IF EXIT THEN 0 OVER father @ !  ( cut )
      DUP left @ OVER rite @ TQ @ ( both subtrees of tq & TQ )
      tq-merge tq-merge TQ! ;        ( paste it back together )
```


Gold Hill delivers GoldWorks.™
GoldWorks.
DESIGNED FOR EXPERT SYSTEM DEVELOPMENT AND DELIVERY

Now you can build and deliver serious expert systems on advanced PCs.

GoldWorks changes the economics of expert system building.

GoldWorks, formerly code-named Acorn, is designed for professional software developers who need to build serious expert systems and integrate them with conventional applications running on 286- and 386-based PCs. It combines the best features of high-end expert system tools with the ability to develop and deliver expert systems *inexpensively,* on advanced PCs.



GoldWorks is the most comprehensive expert system tool available for advanced PCs.

Works like a shell.

GoldWorks gives you the best features of an expert system shell. With the easy-to-use menu interface, you can rapidly prototype and build expert system applications without knowing the underlying programming environment. And you get the GoldWorks tutorial, the San Marco LISP Explorer® tutorial, an on-line help system, and example applications to get you started quickly.

Works like a toolkit.

GoldWorks gives you the best features of an expert system toolkit. You can access the underlying programming environment to extend and customize the system for your specific applications. And you can address up to 15 MB of extended memory on the PC AT (and even more on 386-based PCs).

Works like expert system tools previously available only on high-end workstations . . . at a fraction of the cost.

GoldWorks sets a new standard for expert system development and delivery on advanced PCs. You get frames with multiple inheritance for flexible knowledge representation. Rules supporting integrated forward and backward chaining for powerful inferencing. Object programming for developing modular applications. Plus advanced features for controlling the inferencing process, including rule sets, sponsors, rule priorities, certainty factors, and extensive rule inspecting and debugging facilities. All on conventional hardware—the PC you already use.

Works to develop and deliver your expert systems.

GoldWorks is the only tool that lets you develop and deliver serious expert systems on PC ATs. And **GoldWorks** also takes advantage of PCs based on Intel's powerful 80386 processor, including the COMPAQ DESKPRO 386 and Gold Hill's 386 LISP System.

Works to integrate expert systems with conventional PC applications.

With **GoldWorks**, you can integrate expert system applications with dBASE III and Lotus 1-2-3 . . . integrate C routines and libraries into your expert systems . . . and build and deliver expert systems in network environments.

Works the way you want an expert system builder to work.

GoldWorks from Gold Hill sets the standard by which all other expert system tools will be measured. It was extensively field-tested by developers in dozens of major corporations. And GoldWorks is backed by Gold Hill's comprehensive customer support, training and consulting programs.

Now you have the expert system builder that works the way *you* want to work—**GoldWorks.** To see how it works, order our unique Demonstration Kit, including full color video and complete User's Guide. It's only $49 postpaid, refundable with your **GoldWorks** purchase. To order, call toll-free:

1-800-242-5477.
In Mass., call (617) 492-2071.

**GoldWorks from Gold Hill.
The expert in AI on PCs.**

GOLD HILL™

Gold Hill Computers, Inc.
163 Harvard Street
Cambridge, MA 02139

Two-Bit Analog-to-Digital Conversion

by John Musselman

An example of putting interrupts to work and how software can directly interact with hardware

This article describes a simple technique for measuring an analog voltage that is not only useful but also demonstrates some programming techniques commonly used in real-time applications. The key to such programming is the use of hardware interrupts. A lot of programmers might be wary of hardware interrupts, but once you use them you will find they are a powerful tool. I work with embedded controllers a lot, and so I use them all the time.

Using Hardware Interrupts

The simplest interrupt scheme utilizes a hardware timer to generate an interrupt at regular intervals. How this is done depends on the hardware in the system, so it is not possible to go into much detail. In general, the idea is to set a timer to its free-running mode. In this mode, the timer is set to some value that decrements on each system clock pulse. When the timer reaches zero, it generates an interrupt, is automatically reloaded, and starts to decrement again. Thus, the interrupt routine is entered at regular intervals. This creates a time base for any routines running in the interrupt and guarantees that each routine in the interrupt will be executed within a certain interval. I almost always have one such interrupt running in any system I design. Many routines that might otherwise qualify for their own interrupt can often run in this "master" interrupt, which helps simplify things.

The driver for the analog-to-digital (A-to-D) circuit appears in this kind of interrupt. The frequency of the interrupt is not critical, although the faster it is, the more quickly or more accurately the conversion can be made. Once every millisecond (1,000 times a second) is a convenient figure.

Figure 1, below, is a simplified schematic of the A-to-D converter circuit. Just two bits—one output and one input—interface the circuit to the computer. To understand the circuit, first consider the part of it represented by the output bit, the resistor, and the capacitor. This is actually a simple D-to-A converter. The output bit is set high or low during each interrupt. The resistor and capacitor values are relatively large, so the voltage on the capacitor is an average of the output voltage over time. More 1s and fewer 0s produce a higher voltage; fewer 1s and more 0s give a lower voltage. If the ratio of 1s to 0s is held constant over time, the voltage will be essentially constant and proportional to the percentage of 1 bits being output.

A simple way, then, to program a D-to-A with one output bit, a resistor, and a capacitor would be to repeatedly output m 0s followed by n 1s, where m+n is a constant. This could be programmed with a software counter that decrements on every interrupt. Comparing the counter against some variable determines whether to output a high or a low during that particular interrupt. When the counter reaches 0, it is reset to m+n. The variable thus would control the output voltage as the variable ranges from 0 to m+n.

The A-to-D converter does not use

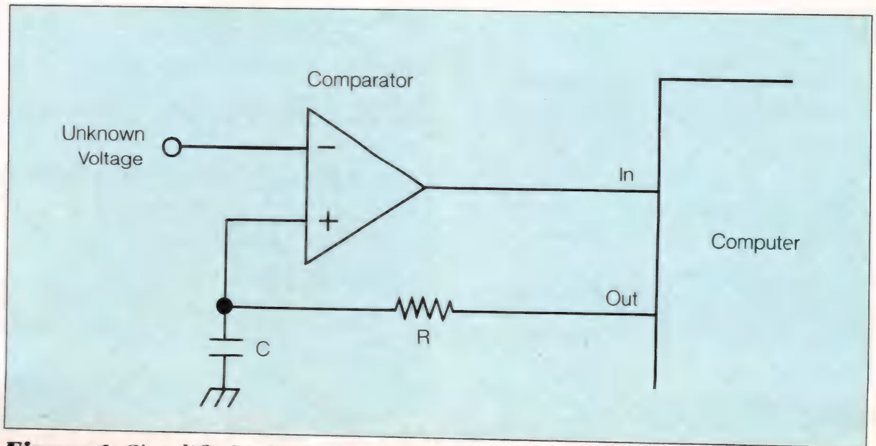


***Figure 1:** Simplified schematic of analog-to-digital converter circuit*

John Musselman, Route 3, Box 344, Escondido, CA 92025. John is VP of engineering for PMC Industries Inc. and is a consulting product design engineer. He has designed numerous microcomputer-based instruments and controllers.

exactly this method to create its output voltage, but the concept is the same. A counter is used to control the conversion cycle, and the ratio of 1 bits to 0 bits output during each cycle determines the output voltage. The difference is that the output is not all 0 bits followed by all 1 bits; the 1s and 0s can appear in any order.

In Figure 1, notice that the unknown analog voltage is fed into one input of the comparator. The other input of the comparator is the voltage generated by the computer, as explained earlier. The output of the comparator is a digital signal that indicates whether the unknown voltage or the generated voltage is higher.

The Software Driver

The A-to-D software driver samples the comparator output at each interrupt. If the generated voltage is lower than the unknown voltage, a 1 is output, charging the capacitor to a slightly higher voltage by the next interrupt. If the generated voltage is higher than the unknown voltage, a 0 is output, discharging the capacitor to a slightly lower voltage. In this way, the generated voltage seeks the level of the unknown voltage and then hovers about it. With large resistor and capacitor values, the searching effect is small and the two voltages are essentially equal. You can determine what voltage you are outputting by counting the percentage of 1 bits; this value is the answer sought.

Example 1, right, shows how you do this. The code is for a TMS7000 series processor. It is fairly easy to read, even if you are not familar with this device, but let me point out a few things. First, the *MOVD* instruction is a 16-bit move. The second byte of the variable is specified as the operand for this instruction, as in *MOVD X+1,Y+1*. Because the processor does not have a 16-bit increment instruction, I have used a 16-bit decrement to count in one's complement. The count is complemented before being saved as the result.

The conversion cycle takes 1,000 1-millisecond interrupts, or 1 second. A 16-bit counter, *COUNT*, keeps track of this time period. Another 16-bit counter, *HIGH*, keeps track of the number of 1 bits output during the cycle. At the end of the cycle, *HIGH* is

```
INBIT:    EQU     1                  ;INPUT BIT POSITION
OUTBIT:   EQU     2                  ;OUTPUT BIT POSITION
          ;
PERIOD:   EQU     1000               ;# OF INTERRUPTS IN A CONVERSION
          ;
          ;
HIGH:     DS      2                  ;COUNT OF HIGH BITS
COUNT:    DS      2                  ;CONVERSION CYCLE COUNTER
RESULT:   DS      2                  ;RESULT OF CONVERSION
          ;
          ;
          ;-----------------------------------------------
          ; INITIALIZATION...
          ;-----------------------------------------------
          ;
          ; MUST BE INCLUDED IN THE SYSTEM
          ; INITIALIZATION BEFORE INTERRUPTS
          ; ENABLED
          ;
          ;
          MOVD    #PERIOD-1,COUNT+1
          ;
          ;
          ;-----------------------------------------------
          ; INTERRUPT ROUTINES...
          ;-----------------------------------------------
          ;
          ; THE FOLLOWING ROUTINES MUST APPEAR
          ; IN AN INTERRUPT WHICH OCCURS
          ; AT REGULAR INTERVALS
          ;
          ;
          ;-----------------------------------------------
          ; I/O...
          ;-----------------------------------------------
          ;
          ;
          ; SEE IF INPUT BIT IS HIGH OR LOW...
          ;
          ;
ATOD:     MOVP    APORT,B
          BTJO    #INBIT,B,HI
          ;
          ;
          ; IF IT'S LOW, GENERATED VOLTAGE IS
          ; BELOW THE UNKNOWN VOLTAGE. OUTPUT A
          ; HIGH.  COUNT ONE MORE HIGH BIT...
          ;
          ;
LO:       ORP     #OUTBIT,APORT
          DECD    HIGH+1             ;NOTE ONES COMPLEMENT COUNT
          JMP     IODONE
          ;
          ;
          ; IF IT'S HIGH, GENERATED VOLTAGE IS
          ; ABOVE THE UNKNOWN VOLTAGE. OUTPUT A
          ; LOW...
          ;
          ;
HI:       ANDP    #255-OUTBIT,APORT
          ;
          ;
IODONE:
          ;
          ;
          ;-----------------------------------------------
          ; CONVERSION CYCLE...
          ;-----------------------------------------------
          ;
          ;
          ; SEE IF CONVERSION CYCLE DONE...
          ;
          ;
          DECD    COUNT+1
          JC      ATODDN
          ;
          ;
          ; IF CONVERSION DONE, SAVE RESULT
          ; AND RESET COUNTERS...
          ;
          ;
          MOVD    HIGH+1,B           ;RESULT IS ONES COMPLEMENT
          COM     A                            ;OF COUNT
          COM     B
          MOVD    B,RESULT+1
          ;
          MOVD    #-1,HIGH+1         ;ONES COMPLEMENT OF ZERO
          MOVD    #PERIOD-1,COUNT+1
          ;
          ;
ATODDN:
```

Example 1: *Analog-to-digital converter driver*

TWO BIT A TO D
(continued from page 23)

dumped to *RESULT*, and both counters are reinitialized. *RESULT* can be read by the main routine at any time it is desired to know the input analog voltage. It updates once a second. All this is transparent to the main routine.

The number generated by the A-to-D routine, *RESULT*, can take any value from 0 to 1,000. This means that there are 1,001 possible values—1 more than the number of interrupts in the conversion cycle. The actual voltage that *RESULT* represents is (*RESULT/1,000*) times the voltage swing of the output bit. If you actually build this circuit, be careful to measure this voltage. A TTL output may not provide the full 5 volts that an NMOS or CMOS output will.

Also, there is nothing magic about the number 1,000. The conversion cycle may consist of any number of interrupts. The more interrupts used in a conversion, the greater the resolution but the longer it takes to get the result. You can choose the number of interrupts to suit the application.

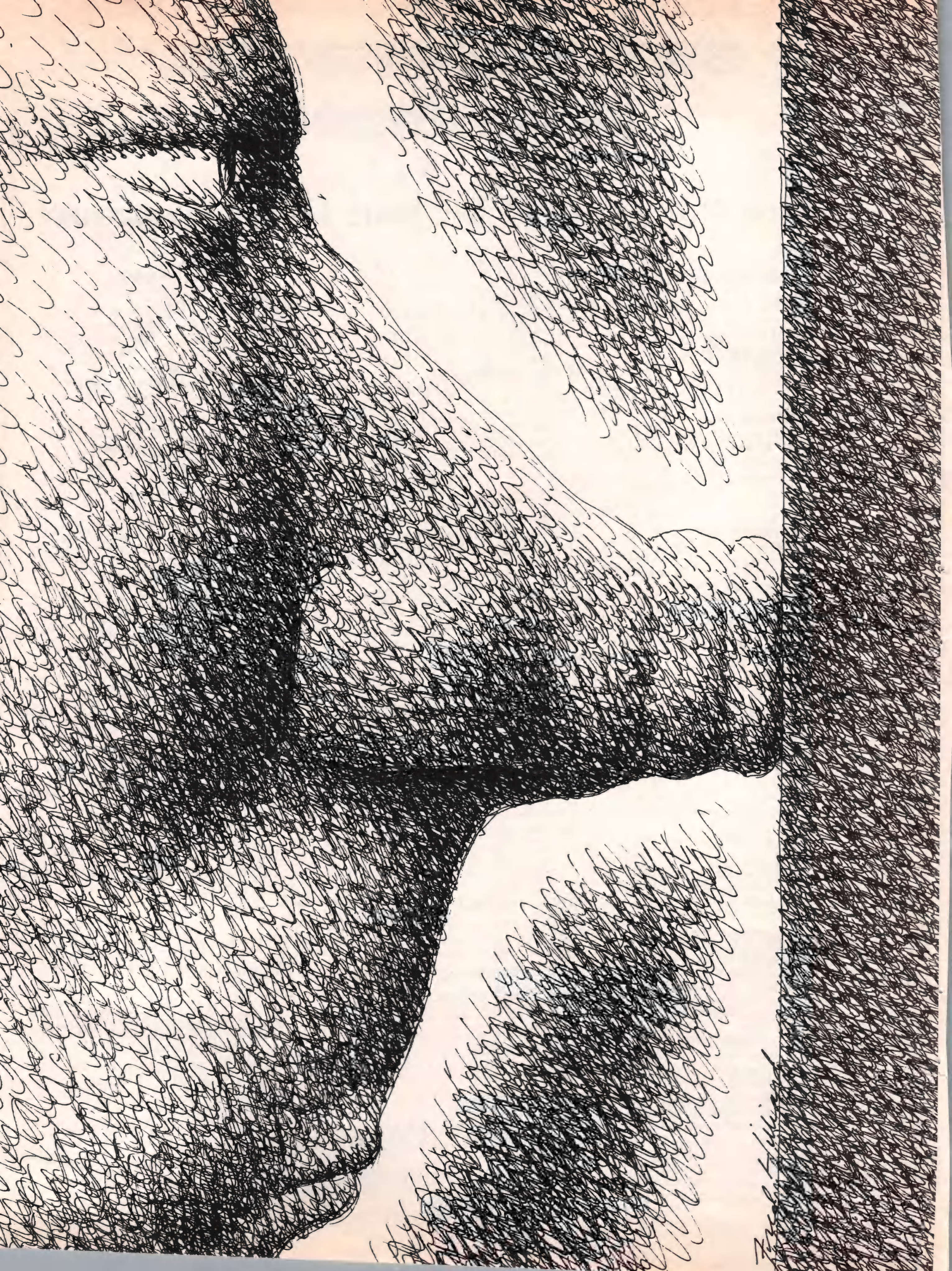
This circuit is a good example of how to put interrupts to work and how software can interact with hardware in a direct manner. Using software in a feedback loop, as in this example, is one of the best ways to convert from the analog to the digital world and vice versa.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal,* 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 3.**



The XOR Chain

by David E. Cortesi

You can construct a doubly linked list with both links in the same word.

In a system in which unsigned binary words and storage addresses are compatible types, a peculiarity of the exclusive-OR function allows both links of a doubly linked list item to be stored in a single word. This space-saving gimmick has been part of the folklore of computing for some time. In this article I explore the mathematical basis of the trick and develop a package of C functions to implement it in a modular style.

The Exclusive-OR Function

Consider the exclusive-OR function (XOR). You can think of XOR as an arithmetic function similar to addition and subtraction; it obeys similar metarules. Like them, for instance, it possesses 0 as an identity element. The identity element of a function is the value whose use preserves an identity over the function. For addition:

A + 0 = A

That is, adding 0 is an identity operation; it leaves A unchanged. In subtraction:

A - 0 = A
A - A = 0

Here, not only does subtraction of 0 leave an identity unchanged, but subtracting identical values yields the identity element. Just so:

A XOR 0 = A (1)
A XOR A = 0 (2)

Relation (2) is the basis of the common assembly-language trick of clearing a register by XORing it with itself.

The addition operation is commutative—that is:

A + B = B + A

This is not true of subtraction, but it is true of XOR:

A XOR B = B XOR A (3)

Addition is also associative—that is:

(A + B) + C = A + (B + C)

Again, this is not true of subtraction but is true of XOR:

A XOR (B XOR C) = (A XOR B) XOR C (4)

The XOR operation differs from addition and subtraction in that all the relations (1) to (4) hold for it, whereas only (1) and (2) are true of subtraction and only (1), (3), and (4) hold for addition. Because all four relations hold for XOR, so does this peculiar rule:

(A XOR B XOR C) XOR B XOR C = A (5)

This may not be immediately obvious. Come at it in steps: apply relations (3) and (4) several times to rearrange the formula so it reads:

A XOR (B XOR B) XOR (C XOR C) = A

By relation (2), this is equivalent to:

A XOR (0) XOR (0) = A

but by (1), 0 XOR 0 is just 0, leaving you with:

A XOR 0 = A

which merely restates (1).

Three Links in a Word

Mathematically, (5) is a tautology, a gassy expansion of (1), but it has a practical use for data storage. From it follows this: if you have a binary word W that contains the result of:

W = (A XOR B XOR C)

and if you know any two of these values independently, you can recover the third value from W. By relation (5):

W XOR B XOR C = A

Similarly, you can recover B knowing A and C, or C knowing A and B. You can use this fact to construct a doubly linked list using but a single binary word to store both links in each list item.

How do you do this? In a doubly linked list, each list item contains the addresses of its predecessor and its successor. Call these A and C, and use the item's own address as B. Set its single link word W to:

W = (A XOR B XOR C)

Now, if a program is examining a list item, it obviously must know that item's address. Provided it arrived at

David E. Cortesi, 415 Cambridge St., #18, Palo Alto, CA 94306. Dave is a former DDJ *columnist.*

the item by tracing the list, it must have come down to it from its predecessor or back from its successor, and hence it must know one of the addresses A or C. Therefore, by (5), it can recover the unknown address from word W and use it to proceed onward in the list.

That's the basic idea. A complete implementation, however, must handle the cases of the empty list and lists of one and two items, should allow for inserting and deleting elements, and ought to be packaged as functional subroutines for simplicity.

Here I'll develop such a package in C. In the following code, I assume that the unsigned binary integer is exactly compatible with a C pointer (as may not be the case in, for instance, the Intel 8086 with a large-memory model), and I've left out the casts that the more picky C compilers may want inserted to make that equivalence manifest. The code favors clarity over absolute efficiency, so you may wish to insert register declarations and other changes for speed.

Defining an Item

In C, the XOR function is denoted by an up arrow and inversion of a Boolean value by comparison to 0. For readability, assume that the following *defines* hold in the example code:

```
#define Xor ↑
#define Not 0==
#define Nil 0
```

Now you can define the structure of a list item:

```
struct Item {
      unsigned link;
            /* other contents of item */
                  };
```

The link word contains the XORed linking addresses. What the "other contents" might be depends on the application, and so I assume that functions to create and destroy *Items* are defined elsewhere:

```
extern struct
      Item *MakeItem( );
extern void
      DropItem(i) struct Item *i;
```

Defining a List

A list consists of a chain of zero or more *Items*, but there has to be an anchor for the chain: a single fixed place where the head and tail of the list can be found. I define its form as *Anchor*:

```
struct Anchor {
      struct Item *head, *tail;
            };
```

For any list there will be just one *Anchor*. Furthermore, I legislate that if a list is empty:

You can think of XOR as an arithmetic function.

```
(head == Nil) && (tail == Nil)
```

Otherwise, *head* points to the first item of the list and *tail* points to the last item. If the list contains but one item:

```
(head == tail) && (head != Nil)
```

You can express these rules in Boolean functions that test an *Anchor*:

```
int ZeroItems(a) struct Anchor *a;
{return(
   (Nil==a->head)&&
   (Nil==a->tail)
          );}

int OneItem(a) struct Anchor *a;
{return(
(a->head==a->tail)
&&(a->head != Nil)
          );}
```

Scanning a List

The *Anchor* of a list gives you access only to the first and last items. If you are to reach intermediate items, you must scan over the list, either forward from the head or backward from the tail.

While scanning the list, your position is always maintained in a pair of pointers that contain the addresses of two items that are logically adjacent in the list. The item that is nearer the head of the list I call the *prior* item; the one closer to the tail I call the *next* item. When I have these items set up with valid addresses, I refer to such a pair of pointers as a *Scan*, because I can use the pair to scan over the list in either direction. Because a pair of pointers is always required, I put them in a record. Because I am always scanning a particular list, I include the address of the list's *Anchor*.

```
struct Scan {
      struct Item *prior, *next;
      struct Anchor *base;
            };
```

Although a list may have only one *Anchor*, it may have any number of *Scans* associated with it. No *Scan* is valid until it has been associated with some list, however. Here's a procedure to do that:

```
void Associate(s,a)
struct Scan *s;
struct Anchor *a;
{ s->base = a; }
```

This procedure is very simple, but in some applications, it might have to do more. It might be desirable, for example, to keep track of the number of scans that are active on a list, and the *Associate* function could do that by incrementing a count in *Anchor*.

A program may scan a list by moving a *Scan* structure from the current position to an adjacent one. But a scan has to start somewhere. This procedure sets a scan to the head of the list:

```
void ToHead(s) struct Scan *s;
{
      s->next = s->a->head;
      s->prior = Nil;
}
```

When a scan is at the head, the *prior* pointer contains *Nil* (nothing precedes the head of a list) and the *next* item is the head item of the list. A simple function can test a *Scan* for this condition:

```
int AtHead(s) struct Scan *s;
{ return( s->prior == Nil ); }
```

XOR CHAIN
(continued from page 29)

A procedure can set a scan to the tail of its list:

```
void ToTail(s) struct Scan *s;
{
     s->prior = s->a->tail;
     s->next = Nil;
}
```

When a scan is at the tail of a list, the *next* pointer contains *Nil* (nothing follows the tail of a list) and the *prior* item is the tail item of the list. A Boolean function can test for this condition:

```
int AtTail(s) struct Scan *s;
{ return( s->next == Nil ); }
```

Recall that a list is empty when the head and tail pointers in its *Anchor* are *Nil*. Verify for yourself that, for a *Scan* associated with an empty list, applying either *ToHead* or *ToTail* makes both *AtHead* and *AtTail* report true. Earlier I defined a function *ZeroItems*, which tests an *Anchor* to see if it's empty. Now I can define a function that tests not an *Anchor* but a *Scan:*

```
int EmptyList(s) struct Scan *s;
{ return( AtHead(s) &&
          AtTail(s)); }
```

Consider the implications for loop control. Either the sequence:

```
ToHead(z);
while (Not AtTail(z))
     {step forward in list}
```

or the sequence:

```
ToTail(z)
while (Not AtHead(z))
     {step backward in list}
```

can work safely—that is, do nothing—on an empty list.

Stepping a Scan

Having set a *Scan* to the head of a list, you want to be able to step it forward in the list. You do that using the identities of XOR. Consider an *Item* that is neither at the head nor the tail of a list. If it is located at address B and if its predecessor is at address A and its successor at address C, then its link word contains A XOR B XOR C. The link word plus the contents of the *Scan* let you step it forward:

```
void GoFwd(s) struct Scan *s;
{ struct Item *i;
     i = s->prior Xor
          s->next Xor
          s->next->link;
     s->prior = s->next;
     s->next = i;
}
```

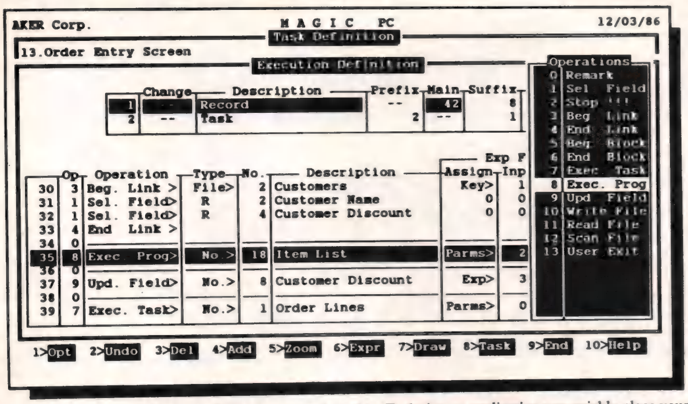
Variable *i* receives the link word of the next *Item* minus the contributions made by its own and its predecessor's address: in short, its successor's address.

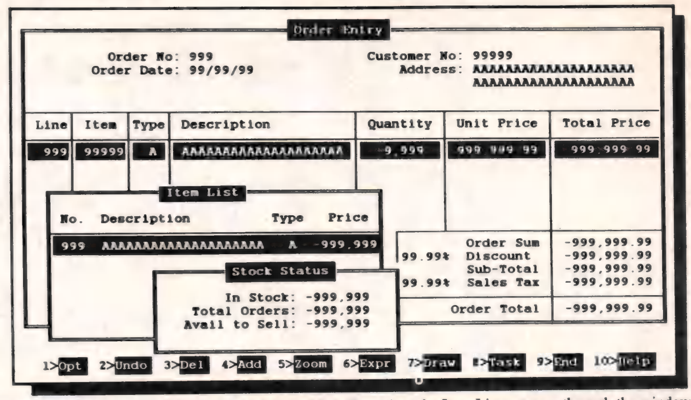
In this way you can process all the items of a list:

```
Associate(z,a);
ToHead(z);
while( Not AtTail(z) )
{
     Process(z->next);
     GoFwd(z);
}
```

If you are accustomed to C, you may prefer to see this written using a *for*

Develop DB applications 10 times faster without the coding pain...you'll swear it's Magic





Visual Programming with Magic PC looks as simple as this. To design an application you quickly place your design specifications into menu-driven Task Tables without having to write a single line of code. For example, just by highlighting the Execute Program operation on the left screen and also highlighting the Item List program in the Program Menu, you tell Magic PC to Zoom into the Item List program through the window shown on the right screen. The window will automatically scroll the Item List data horizontally and vertically, and allow query, "cut and paste" copy or even creation of new Items.

Free yourself from coding

Database professionals throughout the world are discovering a new way to dramatically cut development time.

So can you! With **Magic PC, the Visual Database Language** by Aker.

Consultants, VAR's, Software Houses and DP MIS professionals: If you develop DB applications for a living, now you can tackle any database application 10 times faster than with your DBMS or 4GL.

What makes you so fast with Magic PC?
It's not magic...it's simply because Magic PC finally frees you from coding. And doesn't coding take up most of your time right now?

Magic PC lets you leverage your design skills instead of wasting your time coding. Now you can generate a fully functional prototype in just hours for quick customer feedback, and easily refine the same prototype to a finished application.

All you do is enter your system design specifications directly into Magic PC's non-procedural menu-driven Tables, as ideas come to mind, and Magic PC generates the programs for you automatically.

Magic PC gives you a free hand to design powerful data management systems limited only by your own imagination. Without the time consuming mechanical details of conventional procedural programming. There's your competitive edge. The rest is up to you.

Your biggest time saving comes from Magic PC's dynamic adaptation to spontaneous design changes. You're free to change your design on the fly, and Magic PC automatically updates your programs and data files online. No more time wasted maintaining each program manually with every small change.

Visual Programming Power

You program with Magic PC by describing your data elements with **Data Dictionary Tables** (Files, Fields, Keys), and placing your system design specifications into **Task Description Tables.**

The Tasks can be nested within one another or dynamically Link to satellite Tasks, to give you true One-to-Many relational database power.

Only 13 Task **Operations** harness the power of Magic PC. Operations are specific enough to eliminate the need for tiresome coding, yet elastic enough to produce robust custom applications.

Use the Task building blocks to quickly generate Online Programs: Screens, Window Zooms, Menus; or Batch Programs: Reports, Updates, Data Import/Export and much more.

You develop the Task Tables visually on the screen by highlighting selections from Window Zooms and pop-up menu-driven Tables. You're not forced to follow any particular Table sequence, and there's no coding to slow you down. It's that simple.

You can apply mathematical and logical Expressions, or use the built-in Functions directly in the Task Tables to automate conditional Task processing, to display custom error messages or even invoke external applications such as spreadsheet, word processing or communication programs, transparently from within your Magic PC application programs.

Magic PC generates your application by fusing all your Data Dictionary and Program Tables seamlessly into a single **Integrated Library,** and automatically maintains changes online for optimal, bug-free performance, so you always get it right the first time.

Your application is executed at runtime by a **Magic Run** engine for stand-alone operation, and you can distribute your applications at a low cost and protect your design. Magic Run has a built-in visual interface to manipulate data and get on-the-spot ad-hoc information without any commands or syntax, simply by highlighting selections from menus. Data validation, security and error-checking are done automatically for you without programming.

Magic PC has built-in support for File and Record Locking so you can design multi-user applications for a local area network, and share data with any number of Magic Run users.

Magic PC integrates the Btrieve file manager internally, supporting the B-Tree file structure for fast high performance data access, and fault tolerant recovery during power failures.

Magic PC's powerful **Window Zoom** lets you design composite screens with windows to probe deep into the application through nested windows and manipulate the data underneath. By Zooming from any field, your end-user can conveniently query, copy or even create data in other programs directly through the windows, without stopping their screen session. The window frame size does not limit the available Data View since each window has built-in horizontal and vertical scrolling.

Magic PC is the professional's choice:

IBM France: "IBM encourages Magic PC and salutes such evolution..."

Israeli Air Force: "We were convinced that it was not possible to have a design tool powerful enough to implement real life applications without a programming language. Magic PC changed our mind..."

PC Magazine: "If the thought of programming database applications makes you tremble, Magic PC is for you. The applications generator saves users from the need to deal with much dreaded computer language code..."

PC Tech Journal: "Magic PC is probably the best integrated database application and screen generator that we have seen...very smooth system, and smoothness comes at a premium these days..."

PC World: "Relational data managers and application generators that offer power without programming are a bit like perpetual motion machines—very rare. Into that vacuum comes Magic PC, a data management tool without language that is ideal for turnkey applications..."

PC Week: "Rather than use a written programming language the user is given a great deal of freedom and power to create complex relational database applications...this package is a true time-saver..."

CIRCLE 369 ON READER SERVICE CARD

Get your Magic Tutorial for only $19.95

See for yourself how fast and powerful visual programming can be for you. Order your copy of **Magic PC Tutorial** including Tutorial disks, and a step-by-step tutorial to quickly set up an Order Entry application without coding. All for only $19.95 and We'll credit the Tutorial cost towards the $695 Magic PC price for up to 30 days.

Or save $500 at no risk!

Yes, for a short time only, qualified VAR's can save almost $500 off the list price and get the complete unprotected **Magic PC** software and documentation at a **special VAR price of $199*** with no risk. Keep it at this low price only if it's as good as we say, or return it within 30 days for a full refund if you're not completely satisfied. Act now and you'll also get 2 Magic Run (runtime modules) for only $95.

Order now for immediate delivery

Call this toll-free number now with your credit card or COD charge, or send the Order Coupon below today with your check to Aker.

1-800-345-MAGIC
in CA 714-250-1718

Yes, please rush me the following
Prices include 2nd day shipping

☐ Magic PC Tutorial	$ 25.95
☐ Magic PC (VAR's only)	$209.00
☐ Magic PC	$705.00
☐ Magic Run	$100.00
in CA add 6% tax	$____
Total	$____

Ship to: __________

Address: __________

City/ST/Zip: __________

Phone: __________

OEM inquiries are welcome. Prices valid in North America only.
*Less $19.95 restocking fee. Limit one per customer, subject to availability. Not for resale.
System requirements: IBM PC, XT, AT and 100% compatibles, PC-DOS 2.0 or later, 512K and hard disk.

AKER

Aker Corp. 18007 Skypark Cir. B2, Irvine, CA 92714
Tlx 4931184 AKER UI
Aker S.A. 11 Route de Florissant CH-1206, Geneve
Switzerland Tlx 421792 AKER CH

Trademarks: Magic PC, The Visual Database Language, Window Zoom, Magic Run, Magic LAN and Magic PC Tutorial are trademarks of Aker Corp., IBM PC and PC-DOS are trademarks of IBM Corp., Novell is a trademark of Novell Inc., Btrieve is a trademark of Softcraft Inc.

WINDOWS FOR DATA™

The first choice of professional C programmers

"Windows for Data is the best programming tool I've ever used. It's the most flexible I've seen. Whenever I've wanted to do something, I've been able to find a way."

Steven Weiss,
Stratford Systems

Professionals choose our tools because they are designed, crafted, and supported for professionals. Here at Vermont Creative Software, we understand that performance and pleasure in programming derive from more than a long list of functions. **Windows for Data** provides:

PROFESSIONAL FLEXIBILITY: Our customers repeatedly tell us how they've used WFD in ways we never imagined - but which we anticipated by designing WFD for unprecedented adaptability. Virtually every capability and feature can be modified to meet special needs. You will be amazed at what you can do with WFD.

PROFESSIONAL PERFORMANCE: Screen output is crisp and fast. Windows, menus, and data-entry forms snap up and down from the screen. WFD is built upon and includes **Windows for C, the windowing system rated #1 in speed and overall quality** in PC Tech Journal (William Hunt, July 1985).

PROFESSIONAL RELIABILITY: An unreliable tool is worse than no tool at all. VCS products are known in the industry for their exceptional reliability. Ask anyone who owns one.

PROFESSIONAL DOCUMENTATION: Over 600 pages of documentation provide step-by-step explanations for each major application, a reference page for each function, listings of functions alphabetically and by usage, and a fully cross-referenced index. Extensive tutorials and demonstration programs assist learning.

PROFESSIONAL TECHNICAL SUPPORT: The same expert programmers that develop our products provide prompt, knowledgeable technical support.

PROFESSIONAL PORTABILITY: High-performance versions of VCS products are available for XENIX, UNIX, and VMS, as well as DOS. No royalties on end-user applications.

OUR CHALLENGE AND GUARANTEE

If you have an application where no other tool can do the job, try **Windows for Data.** If it doesn't help you solve your problem, RETURN FOR A FULL REFUND. YOU MUST BE SATISFIED.

Ask for **FREE DEMO DISKETTE**

Vermont Creative Software

21 Elm Ave.
Richford, VT 05476
Telex: 510-601-4160 VCSOFT
Tel.: 802-848-7731

Prices: PCDOS* $395; XENIX, VMS, UNI
*PCDOS specify C compiler.

WINDOWS FOR DATA

for DOS, UNIX, VMS ...

The complete windowing data entry, menu, and help system that does the hard job others can't — we **guarantee** it!

Pop-up data entry windows; field types for all C data types, plus decimals, dates, and times; auto conversion to and from strings for all field types; system and user supplied validation functions; range checking; required, must-fill, and protected fields; free-form movement; multiple-choice field entry; scrollable sub-forms. Branch and nest windows, forms, and menus.

Complete context-sentitive help system with pop-up windows and scrollable text.

Pop-up, pull-down, scrollable, and Lotus-style menus.

NEW FOR DEBUGGING: Exclusive **VCS Error Traceback System** automatically identifies the location and cause of program errors. Eliminates the need to code error checks on all function calls! **VCS Memory Integrity Checking** helps catch those hard-to-detect, memory-corruption errors.

NEW FOR ERROR HANDLING: Install your own error handler to be called whenever a function detects an error.

NEW FORM LAYOUT UTILITY simplifies form design.

CIRCLE 157 ON READER SERVICE CARD

THE ADA® WORLD JUST CHANGED.

Meridian Software Systems, Inc. announces the world's first pre-validated Ada compiler for PC-class machines requiring no additional hardware. The Meridian AdaVantage™ compiler hosted on IBM PC/XT/AT and compatibles running MS-DOS was pre-validated using ACVC version 1.8.

The introductory price is $395.00. After July 1, 1987, the list price will be $795.00.

MERIDIAN
SOFTWARE
SYSTEMS, INC.

23141 Verdugo Drive, Suite 105
Laguna Hills, California 92653
714/380-9800

XOR CHAIN
(continued from page 30)

```
loop:

Associate(z,a);
for(ToHead(z);
      Not AtTail(z);
      GoFwd(z))
            Process(z->next);
```

which amounts to the same thing.

When you process a list from head to tail, the *Items* to be processed are those that turn up in the next position. When you go backward, from tail to head, you take them from the prior slot. Here's a going-backward procedure:

```
void GoBak(s) struct Scan *s;
{ struct Item *i;
      i =s->prior Xor
            s->next Xor
            s->prior->link;
s->next = s->prior;
s->prior = i;
}
```

Here you take the link word of the predecessor *Item* minus the contributions of its own address and of its successor: in short, the address of its predecessor. With this procedure and *ToTail*, you can process a whole list:

```
Associate(z,a);
ToTail(z);
while( Not AtHead(z) )
{
      Process(z->prior);
      GoBak(z);
}
```

It may not be clear that these procedures work for all items of all lists. Before you can be sure they do, you need to define what the link words of the head and tail items contain. The simplest rule works: the link word contains *XOR* 0 for irrelevant addresses. Table 1, below, shows a complete four-item list whose anchor is *Test*. Presume that the following loop is to be executed using the list in this table:

```
Associate(z,Test);
ToHead(z);
while(Not AtTail(z)) GoFwd(z);
```

Trace the contents of *prior* and *next* at each step and assure yourself that all the pointers work out correctly. (Surely you don't take articles such as this one on faith, without at least desk-checking the code?) Then trace them through this loop to verify the rest of the code given so far:

```
for(ToTail(z,Test);
      Not AtHead(z);
            GoBak(z));
```

What would happen if a program applied *GoFwd* just once too often? Or *GoBak*? Clearly you ought to account for these end effects. What shall be done with the head item (which has no predecessor) and the tail item (no successor)? You can either wrap around or stick. Here is a safe *StepFwd* procedure:

Address	Contents
Test.head	A
A->link	A xor B
B->link	A xor B xor C
C->link	B xor C xor D
D->link	C xor D
Test.tail	D

***Table 1:** A four-item list whose anchor is* Test

```
void StepFwd(s) struct Scan *s;
{ if (Not AtTail(s)) GoFwd(s); }
```

that simply sticks when it reaches the tail. Here is a *StepBak* procedure that wraps around:

```
void StepBak(s) struct Scan *s;
{
     if (AtHead(s)) ToTail(s);
     else GoBak(s);
}
```

Inserting an Item

You construct a list by starting with *ZeroItems* true and inserting new items. Example 1, below , shows one way to insert a new item between the prior and next items of a scan. In the statement:

```
s->prior->link Xor=
     (i Xor s->next)
```

you take the predecessor's link, remove from it the contribution of the successor's address (by XORing with *s->next*); and install the value of its new successor—the item with address *i*.

A similar statement fixes the link word in the next item, except that it's the predecessor's address that is removed in favor of *i*. Finally, the new item's link is formed from its own address and that of its new predecessor and successor. It becomes the prior item, so a series of insertions will build up in head-to-tail order.

To verify this, commence with an empty list and four prepared items, as shown in Table 2, below, and execute the following sequence of operations by hand:

```
Associate(z,Test);
Insert(A,z);
Insert(D,z);
GoBak(s);
Insert(B,z);
Insert(C,z);
```

keeping track of the new contents of

```
void Insert(i,s)
struct Item *i;
struct Scan *s;

{
if (AtHead(s))
  s->a->head = i;
else
  s->prior->link Xor=
    (i Xor s->next);

if (AtTail(s))
  s->a->tail = i;
else
  s->next->link Xor=
    (i Xor s->prior);

i->link =
  s->prior Xor s->next Xor i;
s->prior = i;
}
```

Example 1: *A way to insert a new item between the prior and next items of a scan.*

Address	Contents
Test.head	Nil
Test.tail	Nil
A.link	?
B.link	?
C.link	?
D.link	?

Table 2 *An empty list and four prepared items*

SUBSCRIBE & SAVE

$5 SAVINGS

Subscribe to DR. DOBB'S JOURNAL and save over $5—a 15% savings off the cover price!

☐ Please charge my: ☐ Visa ☐ MasterCard ☐ American Express
☐ Payment enclosed ☐ Bill me later

Card # ____________ Exp. date ____________

Signature ____________

Name ____________

Address ____________

City ____________ State ______ Zip ______

ONLY $29.97! YOU SAVE OVER $5.00

Savings based on a full one-year cover price of $35.40. Canada and Mexico add $10 for surface mail per year. All countries add $27 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc. 3448

COMMENTS & SUGGESTIONS

Dear Reader, **June 1987, #128**

Dr. Dobb's has a long tradition of listening to its readers. We like to hear when something really helps or, for that matter, bothers you. In this hectic world of ours, however, it is often difficult to take the time to write a letter. This card provides you with a quick and easy way to correspond and, if you include your name and address, we may use appropriate comments in The Letters column. Simply fill it out and drop it in the mail.

—Ed

Which articles or departments did you enjoy the most this month? Why?

____

____

____

Comments or suggestions: ____________

____

____

____

Name: ____________

Address: ____________

____

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

Dr. Dobb's Journal of
Software Tools
Box 3713
Escondidio, CA 92025-9843

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

Dr. Dobb's Journal of
Software Tools
Box 3713
Escondidio, CA 92025-9843

PLACE
STAMP
HERE

Dr. Dobb's Journal of
Software Tools
501 GALVESTON DR.
REDWOOD CITY, CA 94063

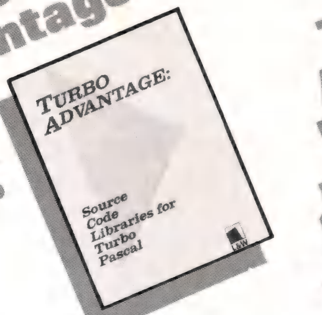
DR. DOBB'S JOURNAL OF SOFTWARE TOOLS

THE R_x FOR PROGRAMMERS

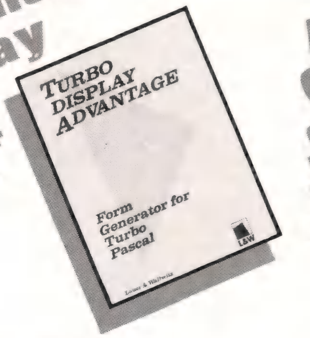
SUBSCRIBE NOW AND SAVE OVER 15% OFF THE NEWSSTAND PRICE!

Turbo Pascal Tools

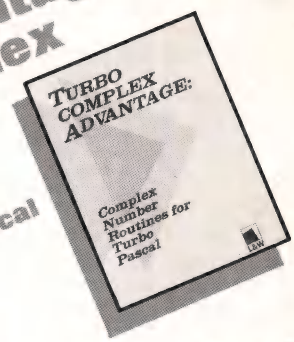
Turbo Advantage
Source Code Libraries for Turbo Pascal



Turbo Advantage Display
Form Generator for Turbo Pascal



Turbo Advantage Complex
Complex Number Routines for Turbo Pascal



This library of more than 220 routines, complete with source code, sample programs and documentation will save you hours developing and optimizing your programs!

Routines are organized and documented under the following categories: bit manipulation, file management, MS-DOS support, sorting, string operations, arithmetic calculations, data compression, differential equations, Fourier analysis and synthesis, matrices and vectors, statistics, and much more! All source code is included.

A detailed manual includes a description of the routine, an explanation of the methods used, the calling sequence, and a simple example. For MS/PC-DOS systems.

Turbo Advantage **Item #070 $49.95**

TURBO Display includes a menu driven form processor, 30 Turbo Pascal procedures and functions to facilitate linking created forms to your program, and full source code and documentation. For MS-DOS systems.

Some of the *TURBO Advantage: Source Code Libraries for Turbo Pascal* routines are necessary to compile *TURBO Display.*

TURBO Display **Item #072 $69.95**

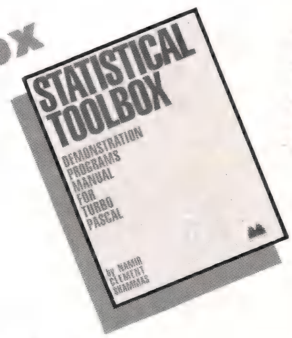
TURBO Complex provides procedures for performing all the arithmetic operations and necessary real functions with complex numbers. Each procedure is based on predefined constants and types. By using these declarations the size of arrays are easily adapted. Each type declaration is a record with both a real and imaginary part. Use these procedures to build more sophisticated functions in your own programs.

TURBO Complex also demonstrates the usage of these procedures in routines for vector and matrix calculation with complex numbers and variables; simultaneous Fourier transforms; calculations of convolution and correlation functions; low-pass, high-pass, band- pass and band-rejection digital filters; and solving linear boundary-value problems.

Source code and documentation is included. For MS-DOS systems. Some of the *TURBO Complex* routines are most effectively used with routines contained in *TURBO Advantage.*

TURBO Complex **Item #071 $89.95**

Stat Toolbox
for Turbo Pascal



Two statistical packages in one!

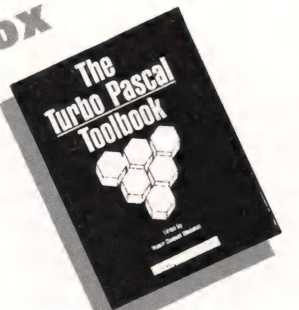
A library disk and reference manual
Use these powerful statistical routines to build your applications. Routines include: • statistical distribution functions • random-number generation • basic descriptive statistics • parametric and non-parametric statistical testing • bivariate linear regression, multiple and polynomial regression.

A demonstration disk and manual
This package incorporates the library of routines into a fully functioning statistical program. Two data management programs are included to facilitate the storage and maintenance of data.

Full source code is included. (For IBM PC's and compatibles. Turbo Pascal version 2.0 or later, and PC-DOS 2.0 or later are required.)

STAT Toolbox **Item #050 $69.95**

Turbo Pascal Toolbox
Edited by Namir Clement Shammas



You'll find:
- an extensive library of low-level routines
- external sorting and searching tools, presenting a new database routine that combines the best features of the B-tree, B+ and B++ trees
- window management, to help you create, sort and overlay windows
- artificial intelligence techniques
- mathematical expression parsers, offering two routines that convert mathematical expressions into RPN tokens
- a smart statistical regression model that searches for the best regression model to represent a given set of data.

All routine libraries and sample programs are on disk for MS-DOS systems, and over 800K of Turbo Pascal source code is included!

Turbo Pascal Toolbook **Item #080 $25.95**
Turbo Pascal Toolbook with disk **Item #081 $45.95**

TO ORDER: Return this coupon with your payment to: M&T Books, 501 Galveston Dr., Redwood City, CA 94063. Or, Call TOLL-FREE 800-533-4372 Mon-Fri 8a.m.-5p.m. In CA call 800-356-2002

YES! Please send me

☐ **TURBO Advantage**	# 070	$49.95	______
☐ **TURBO Advantage Display**	# 072	$69.95	______
☐ **TURBO Advantage Complex**	# 071	$89.95	______
☐ **STAT Toolbox**	# 050	$69.95	______
☐ **Turbo Pascal Toolbox**	# 080	$25.95	______
☐ **with disk**	# 081	$45.95	______

Subtotal ______
CA residents add salestax ____% ______
Add $2.25 per item for shipping ______
TOTAL ______

Name ______
Address ______
City ______
State ______ Zip ______
☐ Check enclosed. Make payable to M&T Publishing.
Please charge my ☐ VISA ☐ M/C ☐ AMEX
Card no. ______
Expiration Date ______
Signature ______

3128 A

XOR CHAIN
(continued from page 35)

Anchor, the *Item* links, and *prior* and *next*.

Deleting an Item

You need a way to delete an item of a list. Deleting comes down to fixing the things that point to an item so they point to other items instead. Once nothing points to it anymore, an item can be discarded. The deletion procedure in Example 2, page 36, deletes the next *Item* of the scan. If there is no next *Item*, it does nothing (ergo, it correctly does nothing to an empty list).

The item to be deleted is *s->next*. The first step is to get the address of its successor, which will take its place. It is obtained from its link field in the manner of *GoFwd*.

Next you repair the predecessor of the deleted item. If it's the head of the list, the anchor must be repaired; otherwise, the predecessor's link field is adjusted to point to the deleted item's successor.

The successor of the deleted item is then repaired. If there isn't one, the deleted item was the tail of the list, and its predecessor becomes the new tail.

You could empty a list with this loop:

```
Associate(z,Test);
ToHead(z);
while (Not AtTail(z)) Delete(z);
```

Although this procedure won't delete when the scan is *AtTail*, the tail item can nevertheless be deleted using:

```
ToTail(z); StepBak(z); Delete(z);
```

Earlier I said that any number of *Scan* structures could be associated with a list. Now that I've defined *Insert* and *Delete*, I should perhaps modify that claim. What might the effect be on one *Scan* if *Insert* were performed via a different *Scan* of the same list? (Answer: not much—the new item might be missed by the other *Scan*.)

What about *Delete*? (Answer: catastrophe–the other *Scan* might step using the link word of a deleted, now-invalid *Item*.)

Conclusion

The C functions shown here form the basis of a package for handling doubly linked lists. These lists have several advantages: because they are doubly linked, they can be traversed in either direction with equal ease, and insertion and deletion are simple and speedy operations. By packaging a list position as a single *Scan* structure, you make it convenient to track several list positions at once, as long as no deletions are allowed.

There are disadvantages, of course. The greatest is that the only practical access to the list is from its ends; it isn't possible to enter it at the middle and then traverse forward or backward. (Traversal requires the addresses of two logically adjacent items, which can't be maintained outside the list itself in the presence of *Insert* or *Delete*.) Thus this list form can't be indexed for quick entry at intermediate points.

Readers of an imaginative turn of mind might explore some other possibilities. For instance, the structure treated here as a linear list could be wrapped around to make an endless ring. In a double-linked ring, *Anchor* might be eliminated in favor of a mandatory single *Item*.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 4.**

```
void Delete(s) struct Scan *s;

} struct Item *i;
if (AtTail(s)) return;
i = s->next->link
  Xor s->prior Xor s->next;

if (AtHead(s))
  s->a->head = i;
else
  s->prior->link Xor= (s->next
Xor i);

if (i==Nil)
  s->a->tail = s->prior;
else
  i->link Xor= (s->next Xor s-
>prior);

DropItem(s->next);
s->next = i;
}
```

Example 2 *A procedure to delete an item in a list*

THE PROGRAMMER'S SHOP

helps save time, money and cut frustrations. Compare, evaluate, and find products.

RECENT DISCOVERY

UI Programmer - Quickly generate dBASE User Interfaces, prototypes. Use supplied templates or create own. Pop-up help, bounce bar menus, screen forms. II, III, FoxBASE+, Quicksilver, Clipper. PC $ 295

AI-Expert System Dev't

Arity Combination Package	PC	$ 979
System - use with C	MS	$ 229
SQL Dev't Package	MS	$ 229
Auto-Intelligence	PC	$ 749
Experteach - Powerful, samples	PC	$ 349
Exsys	PC	$ 309
Runtime System	PC	$ 469
Insight 2+	MS	$ 379
Intelligence/Compiler	PC	$ 749
T.I. - PC Easy	PC	$ 435
Personal Consultant Plus	PC	$2589
Personal Consultant Runtime	PC	$ 85
Turbo Expert-Startup-(400 rules)	PC	$ 129
Corporate (4000 rules)	PC	$ 359

AI-Lisp

Microsoft MuLisp 85	MS	$ 159
PC Scheme LISP - by TI	PC	$ 85
TransLISP - learn fast	MS	$ 89
TransLISP PLUS		
Optional Unlimited Runtime	MS	$ 150
PLUS for MSDOS	MS	$ 179

Others: IQ LISP ($239), IQC LISP ($269)

AI Prolog

APT - Active Prolog Tutor - build applications interactively	PC	$ 65
ARITY Prolog - full, 4 Meg		
Interpreter - debug, C, ASM	PC	$ 229
COMPILER/Interpreter-EXE	PC	$ 569
Standard Prolog	MS	$ 77
MacProlog Complete	MAC	$ 269
MicroProlog - Prof. Entry Level	MS	$ 85
MicroProlog Prof. Comp./Interp.	MS	$ 439
MPROLOG P550	PC	$ 175
Prolog-86 - Learn Fast	MS	$ 89
Prolog-86 Plus - Develop	MS	$ 229
TURBO PROLOG by Borland	PC	$ 69

Editors for Programming

BRIEF Programmer's Editor	PC	Call
EMACS by UniPress Source: $929		$ 299
Epsilon - like EMACS, full C-like language for macros.	PC	$ 149
KEDIT - like XEDIT	PC	$ 99
Lattice Screen Editor - multiwindow, multitasking Amiga $ 89	MS	$ 109
Micro Focus Micro/SPF	PC	$ 139
PC/EDT - macros	PC	$ 229
PC/VI - by Custom Software	MS	$ 109
Personal REXX	PC	$ 99
PMATE - power, multitask	PC	$ 119

Note: All prices subject to change without notice. Mention this ad. Some prices are specials. Ask about COD and POs. Formats: 3" laptop now available, plus 200 others. UPS surface shipping add $3/item.

Compare Products
Use Product Specialists

The Programmer's Shop is much more than the supplier of the largest selection of programmer's software. Trained programming consultants will answer your questions. Ask "product specialists" about Cross Assembler, Translators, Debuggers, or compilers.

Our Services:

- Programmer's Referral List
- Compare Products
- Help find a Publisher
- Evaluation Literature FREE
- BBS - 7 PM to 7 AM 617-826-4086
- Dealers Inquire
- Newsletter
- Rush Order
- Over 700 products
- National Accounts Center

C Language-Compilers

AZTEC C86 - Commercial	PC	$499
C86 PLUS - by CI	MS	Call
Datalight C - fast compile, good code, 4 models, Lattice compatible, Lib source. Dev'rs Kit	PC	$ 77
Datalight Optimum - C	MS	$ 99
with Light Tools by Blaise	PC	$168
Lattice C - from Lattice	MS	$275
Let's C Combo Pack	PC	$ 99
Let's C	PC	$ 59
Microsoft C 4.0- Codeview	MS	$275
Uniware 68000/10/20 Cross Compiler	MS	Call
Rex - C/86 by Systems & Software - standalone ROM	MS	$695
Wizard C	MS	$299
Rom Development Package	MS	$259

C Language-Interpreters

C-terp by Gimpel - full K & R	MS	$229
C Trainer - by Catalytix	PC	$ 89
INSTANT C - Source debug, Edit to Run-3 seconds, .OBJs	MS	$379
Interactive C by IMPACC Assoc.	PC	$209
Run/C Professional	MS	$159
Run/C Lite	MS	$ 89

C Libraries-General

Blackstar C Function Library	PC	$ 79
C Essentials - 200 functions	PC	$ 75
C Function Library	MS	$109
C Tools Plus (1 & 2) - Blaise	PC	$125
C Utilities by Essential	PC	$129
C Worthy Library - Complete, machine independent	MS	$249
Entelekon C Function Library	PC	$119
Entelekon Superfonts for C	PC	$ 45
Greenleaf Functions-portable, ASM		$139
LIGHT TOOLS by Blaise	PC	$ 69

FEATURE

Personal COBOL by MicroFocus - Develop, test, debug, execute ANSI '74 code. Full-screen editor, syntax checker, Animator, forms/screen generator, help. Compatible with Level II. PC $169

RECENT DISCOVERY

NET-TOOLS - Access NETBIOS-compatible network systems from Microsoft C, Pascal, FORTRAN, Assembler, Lattice C. Full Source, No Royalties. PC $129

dBASE Language

Clipper compiler	PC	Call
dBASE II	MS	$329
dBase III Plus	PC	$429
dBASE III LanPack	PC	$649
DBXL Interpreter by Word Tech	PC	$139
FoxBASE+ - single user	MS	$349
Quick Silver by Word Tech	PC	$499

dBASE Support

dBase Tools for C	PC	$ 65
dBrief with Brief	PC	Call
DBC ISAM by Lattice	MS	Call
dBx Translator to C	MS	$319
dFlow - flowchart, xref	MS	Call
Documentor - dFlow superset	MS	Call
Genifer by Bytel-code generator	MS	$299
QuickCode III Plus	MS	$249

Fortran & Supporting

50:More FORTRAN	PC	$ 99
ACS Time Series	MS	$399
Forlib+ by Alpha	MS	$ 59
MS Fortran - 4.0, full '77	MS	$279
No Limit - Fortran Scientific	PC	$115
PC-Fortran Tools - xref, pprint	PC	$179
RM/Fortran	MS	Call
Scientific Subroutines - Matrix	MS	$139

Multilanguage Support

BTRIEVE ISAM	MS	$185
BTRIEVE/N-multiuser	MS	$455
Flash-Up Windows	PC	$ 79
GSS Graphics Dev't Toolkit	PC	$375
HALO Graphics	PC	$209
Development Package	MS	$395
Informix - by RDS	PC	$639
Informix 4GL-application builder	PC	$789
Informix SQL - ANSI standard	PC	$639
Opt Tech Sort - sort, merge	MS	$ 99
PANEL	MS	$215
Pfinish - by Phoenix	MS	$229
PolyLibrarian by Polytron	MS	$ 79
PolyBoost - speed I/O, keyboard	PC	$ 69
PVCS Corporate-source control	MS	$319
QMake by Quilt Co.	MS	$ 85
Report Option - for Xtrieve	MS	$109
Screen Sculptor	PC	$ 95
SRMS - source control	MS	$109
VXM - multi-env. link	MS	$195
Xtrieve - organize database	MS	$199
ZAP Communications - VT 100	PC	$ 89

FEATURE

Turbo C by Borland. ANSI Compiler supports 6 models including tiny and huge, has floating point, interactive editor, Make. Speed development. PC $ 69

We support MSDOS (not just compatibles), PCDOS, Xenix-86, CPM-80, Macintosh, Atari ST, and Amiga.

VISA **HOURS** MasterCard

8:30 AM-8:00 PM EST.

Call for a catalog, literature, advice and service you can trust

"I have been pleased with your catalog selection, the knowledge of your telephone answering staff, and the promptness of your service. I have every wish to become a continuing customer of The Programmer's Shop.

Carl C. Rollo

CIRCLE 133 ON READER SERVICE CARD

THE PROGRAMMER'S SHOP

provides complete information, advice, guarantees and every product for Microcomputer Programming.

AUTO-INTELLIGENCE

The First Automatic Knowledge Acquisition System

List: $990 Our: $749

IntelligenceWare, Inc.

9800 S. Sepulveda Blvd. Suite 730
Los Angeles, CA 90045
Telephone: (213) 417-8896; Fax (213) 417-8897

CIRCLE 301 ON READER SERVICE CARD

WANT TO ADD

WINDOWS, ICONS, FONTS, FAST GRAPHICS, DIALOG BOXES, PROCESS MANAGEMENT, AND DEVICE INDEPENDENCE

TO YOUR IBM PC PROGRAMS?

If you have ever wished that you could develop stunning Macintosh-like programs on the IBM PC without the overhead of an enormous operating environment like Windows or GEM, then you need the **SYNERGY DEVELOPMENT TOOLKIT, from Matrix Software.**

Using a memory resident runtime module only **20K** in size (versus as much as 300K for Windows), you can develop applications with: overlapped and tiled windows; pull-down menus with half intensity options and automatic sizing; fast graphics function calls to draw shapes, lines, boxes, and create intricate fill patterns in both regular and irregular areas; have full device independence (drivers for numerous devices, including CGA, EGA and Hercules are included); sophisticated text input and output, with fonts in different styles and sizes; full keyboard support (your programs won't need a mouse!) and powerful mouse support; and process management calls to efficiently manipulate system resources.

The Toolkit has gateways to support the following languages:

- Turbo Pascal
- IBM/MS Pascal
- Microsoft & Lattice C
- Macro Assembler
- Basic
- dBASE II/III Compilers

In addition, the Toolkit includes a powerful collection of tools including a graphics resource editor for developing your own icons and fonts.

NEW! The Toolkit also **includes a free copy of Synergy Layout**, a revolutionary software development tool that dramatically increases your productivity by actually generating bug-free source code in Macro, C, and Turbo Pascal.

For further information, contact Matrix Software at (617) 567-0037.

List: $395
Our: $349

CIRCLE 302 ON READER SERVICE CARD

MUMPS

Now quickly and easily develop multi-tasking applications on your PC!

Now get the speed, power and flexibility of MUMPS in a truly concurrent processing environment with COMP Computing Standard MUMPS (CCSM).

CCSM provides the programmer with these benefits:

* Save time and money by writing Applications in 1/3 to 1/5 the amount of code.
* Fast data access with B-Tree File Structure and Automatic disk caching.
* Unlimited program size and variable space with advanced virtual memory system.
* 100% portable from micros to minis to mainframes.
* Easily write multi-tasking and multiuser applications.

And with MUMPS, all you need to do to start a background process is: "JOB ^ROUTINE". No memory allocation, no table set-up, no hassles; the system handles everything.

CCSM is also available in a multi-tasking version for the Macintosh.

Multi-Tasking:	List - $149.95	Ours - $129
Multiuser:	List - $450.00	Ours - $369

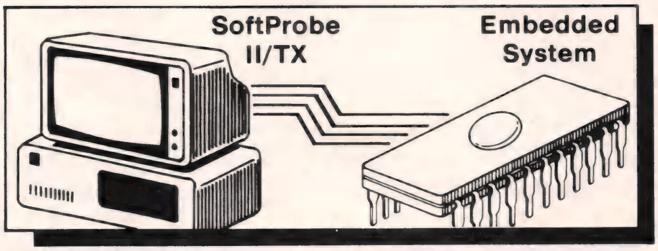
For a demonstration diskette of CCSM, send two dollars to the address below.

MGlobal
1601 Westheimer, Suite 201
Houston, Texas 77006

1-800-257-8052
In Texas: 1-713-529-2576

CIRCLE 303 ON READER SERVICE CARD

SoftProbe II/TX, a High-Level Solution for Embedded System Debugging.



FEATURES

- Source Level Debug
- Supports High Level Language Data Types
- C-like Command Language with Expression Evaluation and Flow Control
- Displays Program Execution Trace in High Level and Assembly Languages
- Friendly User Interface with Macros and On-line Help

TARGET SYSTEM

- Any iAPX-86/186 Based System with a USART
- IBM PC or Compatible.

BENEFITS

- Debug on Actual Target System
- Debug at High Level Language Level
- Mixed Language Debugging Facilitates Firmware Testing

LANGUAGE SUPPORT

- C
- PL/M
- ASM

SYSTEMS & SOFTWARE INC
3303 Harbor Blvd.,
C-11, Costa Mesa, CA 92626
(714) 241-8650 FAX (714) 241-0377

List: $750 Ours: $695

SoftProbe is a registered trademark of Systems & Software, Inc.
IBM PC is a registered trademark of International Business Machines Corp.

Call Today for FREE detailed information or try Risk-Free for 31 days.

800-421-8006
HOURS: 8:30 A.M. - 8:00 P.M. E.S.T.

THE PROGRAMMER'S SHOP™
Your complete source for software, services and answers
5-D Pond Park Road, Hingham, MA 02043
Mass: 800-442-8070 or 617-740-2510 2/87

An Extended IBM PC COM Port Driver

by Thomas A. Zimniewicz

The IBM PC and its compatibles support two serial ports—COM1 and COM2—in a wide variety of configurations. Four basic functions are provided—data in, data out, set configuration, and read status. What the IBM PC does not do is walk and chew gum at the same time. If your disk is seeking and more than one character arrives, it is lost. This article describes what I think a COM port driver should do, discusses some design rules and implementation pitfalls of device drivers, and presents a device driver modification with the following features:

- buffered input—no lost data
- input flow control—XON/XOFF or control lines
- output flow control—control lines
- works with existing software
- removal—without a reboot
- 19,200- and 38,400-baud operation
- type-ahead with CRTs
- upward compatible

What's Wrong?

During the first part of my career as a software engineer, I was spoiled by minicomputer operating systems that were good at capturing serial data. More recently, I had the pleasure of working with Unix—a pleasure, that is, in all aspects except capturing serial data. My introduction to Unix was on a PDP-11/44 that could miss characters from serial data input at 300 baud on a busy day. Every time my job required monitoring serial data, it was a bad day. Then one

> ***What the IBM PC does not do is walk and chew gum at the same time.***

day I noticed an idle IBM PC that had nothing to do (no other users to get in my way) except perhaps collect my data. It would be simple—just *getchar* from COM1 and *putchar* to disk, and with no other work, the IBM PC could certainly handle at least 9,600 baud. A while later it was working, but (and there always is one) it seemed that there were gaps in the data. (I discovered later that the gaps coincided with the flashing of the disk light.) I gave up. Much later I found myself using an IBM PC a little. I didn't like the keyboard and monitor, so I hooked up my CRT to a COM port and then found there was no type-ahead. I got mad enough to fix it.

So, what were my goals? The main thing was buffered interrupt-driven input to allow type-ahead and high-speed serial data capture. I wanted to use these features with existing programs, too. This meant the solution had to be fully compatible and somehow get between programs and the hardware. I had to modify PC-DOS and/or the BIOS in a way that was transparent to an application-level program. In this sea of bad news, there was one very bright light—everything that PC-DOS and the BIOS lacked in capability, it made up for in flexibility. The hooks were there, device drivers could be installed, interrupts could be stolen. If the new COM port software were installed on an IBM PC, then all well-behaved programs (those that used PC-DOS or the BIOS instead of directly accessing the hardware) would automatically gain the new features. Several methods of input/output flow control were possible and the hardware could handle higher baud rates . . . I sure do get carried away easily.

Can It Be Fixed?

The IBM PC actually has two levels of software to handle the COM ports. The high-level code is part of PC-DOS and is loaded into RAM during the boot-up of PC-DOS. This code handles those functions that do not depend on the details of the hardware and provides an interface to the COM port. This level is called a device driver and is accessed by high-level PC-DOS operations such as those that open and read serial data system calls. The low-level code is hardware specific and is contained in ROM. This code is specific to the operation of the COM port hardware. This level is called the BIOS and is called by the device driver to perform operations specific to a COM port, such as setting the baud rate or getting a character from the port hardware.

When I started to investigate the job at hand, I suspected that a whole new device driver would be required. The good news was that PC-DOS faithfully used the BIOS for all COM port operations. This meant that all I needed to do was to write a replacement for the parts of the BIOS that handle the COM port. For a while the job looked really easy. I considered stealing the software interrupt

Thomas A. Zimniewicz, 2695 Pond Rd., Lima, NY 14485. Thomas is a software consultant.

(int 14h) that is used to access the BIOS COM port code, going to my code for operations that needed changing, and otherwise just passing the job off to the BIOS. Unfortunately, I had to abandon this easy way out because of negative side effects caused by almost every BIOS function. For example, the BIOS "send a character" function sets the control signals in ways that are incompatible with some of the required flow control modes.

To illustrate some of the technical issues of handling a COM port, let's take a simplified look at the job of reading serial data. When the BIOS call is made to read a character, it loops until a character arrives at the port or a counter expires. Either the character or a time-out status is returned. Most programs and PC-DOS ignore the time-out and try again. The COM port hardware has only a single character buffer. If the buffer in the COM port hardware overflows while the processor is doing anything else, it is lost. The solution to these lost characters is to have the COM port generate an interrupt when a character arrives and have the interrupt handler save the character in a buffer until the program gets around to needing it. The code might look like this:

```
COM port interrupt handler
  read character
  if buffer count < buffer size
    put character in the buffer
    increment buffer count
  else
    set overrun error

read character routine
  loop
    if buffer count > 0
      decrement buffer count
      take character and return it
    if timer expired
      return time-out status
  endloop
```

If this algorithm were implemented, it would probably pass all its initial tests. Then, when you least expected it, your input would be garbled and no errors would be reported. What you have here is your classic race condition. Imagine the buffer is exactly full when the read routine is called and the count is decremented to indicate room in the buffer. Immediately after the count is decremented but before the character is removed, an interrupt occurs. The handler puts another character in the buffer that is really still full. Things go downhill from here. This type of error is an easy trap for beginners (and sometimes a big embarrassment for pros). One possible fix is to disable interrupts during critical portions of the read character routine. (Note that COM port interrupts are automatically disabled by the hardware during the processing of a COM port interrupt; there is no need to worry about reentering the handler itself). This works, but it has serious drawbacks. If interrupts are off for too long, serial data (and other real-time events) can be lost. The best solution is to order the events in the read character routine so that an interrupt cannot hurt it. If interrupts must be disabled, try to limit the duration. To fix the bug above, just take the character from the buffer before decrementing the buffer count.

I wanted buffered input to allow type-ahead and high-speed serial data capture.

Code Walk-Through

Excom is functionally equivalent to the BIOS *int 14h* COM port handler with three significant enhancements—it provides interrupt-driven buffered input and has an extended set of configuration parameters, including flow-control selection and higher baud rates. Excom is made up of three major sections—the COM port data input interrupt handler, the replacement for the BIOS *int 14h* handler, and the code to install and initialize excom. A problem exists in trying to describe excom, unlike a well-written application in which functions can be isolated to a single routine or group of routines, because its structure dictates that many of its capabilities are distributed throughout the code. I will first take you on a quick trip through the listing (Listing One, page 64) to show its structure and then later describe several of excom's more interesting functions.

Excom is a .COM program that is run once before the COM ports are needed and that uses the terminate-and-stay-resident feature of PC-DOS to remain active as long as is needed. If required it can be removed, restoring the interrupt vectors to their previous values and freeing memory. Two requirements for a .COM program are to have all segment registers pointing to the start of the program's memory space and to have an executable origin of *100h* to make room for the program header. Execution starts at location *100h*, which is a jump to the initialization code. A few constants are then defined, the first three of which have to do with the buffer size and can be modified to tune the excom buffers to your needs. The Microsoft assembler supports a concept similar to structures in C. A structure called *pcb* (port control block) is defined to describe the dynamic state of a port. It is defined as a structure so that one piece of code can handle two ports just by setting a register to point to the appropriate structure. Storage for two port control blocks is then allocated. Space is reserved to store the old interrupt vectors required for the program to remove itself. Next the baud rate table contains "magic" numbers required to set the UART's baud rate clocks. Note that the last two entries are used for the excom extended baud rates.

The first executable code is the COM port hardware interrupt handler. The label *int0B* is the entry point for *int 0Bh*, COM2, whereas *int0C* is for *int 0Ch*, COM1. Each entry sets *ds:si* to point to the *pcb* associated with its port and enters common code. The rest of the routine reads the character, puts it in the buffer, takes any action required for flow control, and exits. There is no analogy to this code in the BIOS. The second code segment is the *int 14h* handler, which performs five distinct functions corresponding to the value in the register *ah*. The functions are initialize port, transmit a character, receive a character, get port status, and extend initialization. At the

COM PORT DRIVER
(continued from page 43)

end of excom is the initialization. This is the code that runs only when excom is being installed. Initialization starts by releasing the memory for excom's copy of the environment, which is not used. The body of the initialization uses PC-DOS calls to get the old interrupt vectors and then install the three new interrupt handlers. The interrupts that are thus stolen are *0Bh* (COM1 hardware), *0Ch* (COM2 hardware), and *14h* (BIOS COM port calls). Then the excom COM port data structures are initialized. This initialization code is at the end so that it is not kept as part of the resident code. Initialization ends with the PC-DOS call to make everything before the initialization code stay resident.

Key Functions

The BIOS does COM port input by waiting for a character to arrive. Even though it is not used, the IBM PC has all the hardware required to generate an interrupt when a character arrives. This capability is activated by changing the interrupt vector (a RAM location) to point to the appropriate handler (see lines 555–592 of Listing One). Now the vectors are set up, but because the BIOS doesn't use COM port interrupts, the hardware has not been initialized to generate interrupts. Two separate pieces of hardware have to be initialized—first the COM port hardware (an 8250 UART) has to be programmed to generate interrupts and then the interrupt controller (an 8259) has to be set to allow the interrupt to be sent on to the CPU. I expected this to be a simple flip through the data sheets and a dozen lines of code. It didn't work—no interrupts were generated. After much pain, I discovered that a spare output called OUT2 on the UART was used to gate the interrupt signal from the UART. I have no idea why. I then asserted OUT2 on the UART and *voilà*—interrupts (see lines 290–295). When an interrupt does occur, the CPU stops what it's doing and jumps through the appropriate vector—in this case to *int0B* or *int0C* (lines 73–158). The interrupt handler then reads the character, which resets the interrupt status in the UART and

sends an interrupt complete command to the interrupt controller before returning.

The BIOS port status request simply reads the two UART status registers and returns them to the caller. One register is called the line status and contains bits for receive errors, receive data ready, time-out, and transmit ready. The other is called modem status and has the UART's control lines. When characters are transmitted or received, the *ah* register returns the line status. When a port is initialized or status is requested, *ah* is returned as above and *al* returns the modem status. In excom input characters are stored in a circular buffer as they are read from the UART and are then removed from the buffer when a character read takes place. This presents a dilemma for excom because some status bits are associated with the received character whereas others are really UART status. The solution is to have the interrupt handler read the status associated with receive characters and buffer both character and status together. Then when a status is needed and there are characters in the buffer, the status associated with input characters is taken from the next character in the buffer. The rest of the status is taken directly from the UART (see lines 429–457). This approach is not always perfect, but excom has been used successfully with many port handling packages, including COMMAND.COM.

One of the major goals for excom was to provide configurable flow control for input data. The objective of flow control is to signal the source of the data to stop sending when the buffer is getting full (see lines 96–120). Then, when the buffer is emptied a bit, the sender is told to resume sending (see lines 394–409). The exact number of characters in the buffer when input is stopped and started is configurable (lines 16–18). Having the stop and restart values be different by a few characters tends to minimize the number of times the data is stopped and restarted. The actual mechanisms used to tell the sender to stop are any combination of DTR (data transfer register), RTS (request to send), and XON/XOFF (Control-S/Control-Q). DTR and RTS are turned off as the buffer fills and turned back on when room becomes available. Control-S is transmitted as the buffer fills, and Control-Q is transmitted when room becomes available. I always thought that RTS was used as described by its name, but I've seen others use it for input flow control. It was easy to add and it didn't seem to do any harm, so I used it for input flow control, too. Add to this arsenal the ability to cross a few wires in a cable, and almost any situation is covered—except, of course, the sender that ignores your attempts. This is where excom shines—interrupts and big buffers provide great power. Data can be copied from a COM port to a floppy on a lowly old 4.77-MHz IBM PC at 9,600 baud without missing a character. To complete the picture, I also added output flow control. The BIOS requires that both DSR and CTS be set before a character can be sent. Excom can be configured to ignore either or both of these signals.

In order to be useful, excom had to be upward compatible with the BIOS *int 14h*. The area of extended initialization options presented some special problems. There are no unused bits in the BIOS set configuration command. Therefore I had to add a new command—*int 14h* with *ah* having the values from 0 to 3 were already used, so *ah* = 4 was the obvious choice (lines 460–499). The problem was that the old initialization—*ah* = 0—included the baud rate and the new initialization could also specify baud rates not included in the BIOS. The obvious solution was to require that the BIOS initialization occur before the extended initialization. If excom was to be used with existing software, though, this restriction was impractical. The actual solution was to have excom ignore the baud rate in the BIOS initialization if the extended initialization had been used to set a nonstandard baud rate (lines 304–306). Thus, you set the baud rate to 19,200, start your existing software, and a BIOS initialization call is made, but excom ignores the baud rate. Another concession to upward compatibility is that the extended options are relative to the BIOS defaults. Excom allows DTR input flow control to be enabled, whereas output CTS flow control can be disabled.

(continued from page 45)

Not All Is Gold

Along the way from the problem to the solution, there were many little struggles, one of which may be of interest. The rest are too embarrassing to put into print.

While first using excom, I ran into a strange problem. I have a homemade shell that reads the keyboard (CRT hooked to COM port) by calling *int 14h* directly. I ran a program that did CRT output via PC-DOS but did not read any keyboard characters. While it was running, I typed a command ahead. When the program stopped, my type-ahead was echoed minus the first character. This was early in my use of excom, so I assumed a bug in excom. Ignoring the problem, I continued what I was doing. Some time later I ran an interactive program that used PC-DOS for keyboard inputs, and my lost character from 20 minutes ago appeared on the screen as the first character echoed. Wow! At first I passed it off as a bug in excom's input buffer and a coincidence. It happened again! I considered a career as a crossing guard.

You may have noticed that I described the exact way in which the above programs read their input—that was the key to understanding. PC-DOS makes a feeble attempt to do some input buffering, and when PC-DOS calls *int 14h* to transmit a character to the port, a status is returned. If a data ready is indicated, PC-DOS does a data read and saves the character for later. PC-DOS only stores one character in this way as far as I can tell. So my type-ahead was eaten by PC-DOS, to be provided to the next call to PC-DOS for a character. Beware of mixing PC-DOS and BIOS character reads!

Anything Left to Do?

Of course there is! The two questions I've heard most often as a software engineer are "When will the code be complete?" and "How many more bugs are there?" The answers have always been the same: "About a week or two" and "Three." The answers are the same here.

If you put a file on a floppy and enter the command *type file*, you will notice a flip-flop of data scrolling by and disk activity. This is because the BIOS waits for both the disk and the display. Excom has input buffering to prevent loss of data. Output buffering won't fix any problems, but it will speed up data output in the case when the source of the data is a slow device. If excom had output buffering, it would take the characters faster than they were actually being transmitted. Thus PC-DOS would think a block of data had been sent and go to the source for more while excom was still sending data out of its buffer. The flow would be limited by making PC-DOS wait when excom's output buffers were full.

Earlier I mentioned that excom can work with any well-behaved program. What about the naughty ones? At work, I use an old version of CrossTalk that steals the COM port interrupt and doesn't put it back. I don't know if this has been fixed. Are you listening CrossTalk folks? Upon return from CrossTalk, excom has lost the COM port interrupt. One possible solution would be to run CrossTalk from a batch file that removed excom, ran CrossTalk, and then reinstalled excom. The problem with this is that you must identify and fix all offenders. It may be possible for excom to examine its environment for damage and perform repairs. I'm sure this would not cover all such problems, but I think all of them that I am currently aware of could be fixed. Self-repairing programs—maybe I should do an article on "Core Wars."

An interesting possibility with excom is nesting several invocations.

When I use my shell on a COM port with or without excom installed, XON/XOFF output flow control seems to work. There must be problems. I have heard of requests to provide XON/XOFF in relation to printer drivers and so on. In any case, because PC-DOS may not see characters until some time after excom does, any PC-DOS attempt at XON/XOFF output flow control is hampered by excom. This implies that XON/XOFF output flow control should be added to excom.

Because the IBM PC does not have memory management hardware, having a memory-resident program occupy the wrong location can cause problems. When a shell runs a program, it is loaded just after the shell. If the program remains resident upon its termination, the shell is in a squeeze. The shell cannot get more memory to hold shell variables, environment, temporary buffers, and so on. A desirable extension to excom would be to have it relocate itself to the end of RAM and remain resident there.

Excom has a bug! Software is no fun if its perfect. Control-S and Control-Q are sent without any regard to the UART being ready or the control lines being correct. This sounds pretty bad, but in reality it's only a problem when there is a heavy flow of data in both directions, which is pretty rare. The reason I allow this bug to exist is that the proper solution requires interrupt-driven output. *DDJ* has only so many pages for my giant listings.

The version of excom on the listings disk is an updated version with interrupt-driven output, corrected sending of Control-S/Control-Q, the ability to repair the damage done by ill-behaved programs, the ability to do output XON/XOFF flow control, and self-relocation to the end of RAM. (See the "Availability" section at the end of this article.)

Use Excom on Your IBM PC

To make excom.com, use the followings commands:

```
masm excom.asm ;
link excom.obj ;
exe2bin excom.exe excom.com
del excom.exe
```

Normally, you should install excom early in the boot process—before a *print /d*—because the print spooler steals *int 14h*. The real-time clock initialization program on my clone messes up the *int 14h* vector for an unknown reason; therefore I in-

"How to protect your software by letting people copy it."

By Dick Erett, President of Software Security

Inventor and entrepreneur, Dick Erett, explains his company's view on the protection of intellectual property.

"A crucial point that even sophisticated software development companies and the trade press seem to be missing or ignoring is this:

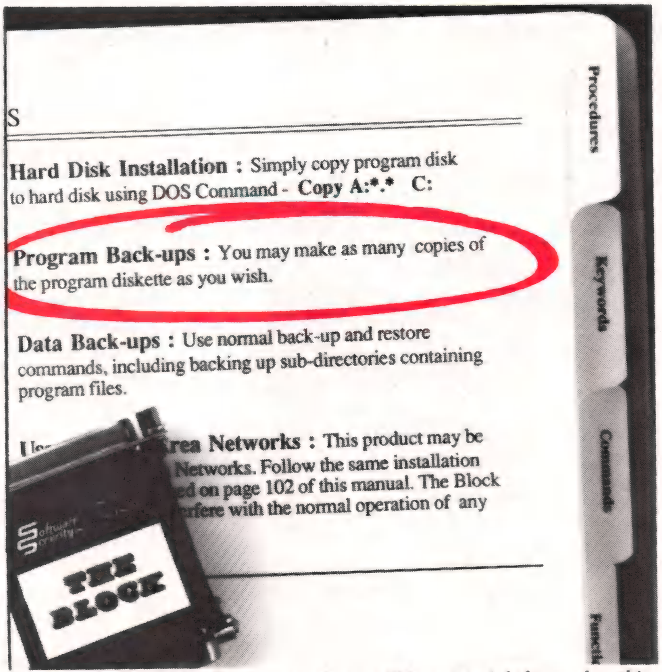
Software protection must be understood to be a distinctively different concept from that commonly referred to as copy protection.

Fundamentally, software protection involves devising a method that prevents unauthorized use of a program, without restricting a legitimate user from making any number of additional copies or preventing program operation via hard disk or LANs.

Logic dictates that magnetic media can no more protect itself from misuse than a padlock can lock itself.

Software protection must reside outside the actual storage media. The technique can then be made as tamper proof as deemed necessary. If one is clever enough, patent law can be brought to bear on the method.

Software protection is at a crossroads and the choices are clear. You can give product away to a segment of the market, or take a stand against the theft of your intellectual property.



Soon all software installation procedures will be as straightforward as this. The only difference will be whether you include the option to steal your product or not.

"...giving your software away is fine..."

We strongly believe that giving your software away is fine, if you make the decision to do so. However, if the public's sense of ethics is determining company policy, then you are no longer in control.

We have patented a device that protects your software while allowing unlimited archival copies and uninhibited use of hard disks and LANs. The name of this product is The BLOCK™.

The BLOCK is the only patented method we know of to protect your investment. It answers all the complaints of reasonable people concerning software protection.

In reality, the only people who could object are those who would like the option of stealing your company's product.

"...eliminating the rationale for copy-busting..."

Since The BLOCK allows a user to make unlimited archival copies the rationale for copy-busting programs is eliminated.

The BLOCK is fully protected by federal patent law rather than the less effective copyright statutes. The law clearly prohibits the production of work-alike devices to replace The BLOCK.

The BLOCK attaches to any communications port of virtually any microcomputer. It comes with a unique customer product number programmed into the circuit.

The BLOCK is transparent to any device attached to the port. Once it is in place users are essentially unaware of its presence. The BLOCK may be daisy-chained to provide security for more than one software package.

Each software developer devises their own procedure for accessing The BLOCK to confirm a legitimate user. If it is not present, then the program can take appropriate action.

"...possibilities... limited only by your imagination..."

The elegance of The BLOCK lies in its simplicity. Once you understand the principle of The BLOCK, hundreds of possibilities will manifest themselves, limited only by your imagination.

Your efforts, investments and intellectual property belong to you, and you have an obligation to protect them. Let us help you safeguard what's rightfully yours. Call today for our brochure, or a demo unit."

Software Security inc.

870 High Ridge Road Stamford, Connecticut 06905
203 329 8870

PAINLESS WINDOWS.

Windows. Data Entry. Menus. Finally, a C programmers' tool that makes them as easy to use as *printf()*. With Greenleaf DataWindows™, you move in quantum leaps!

Snazzy Window Treatments

DataWindows represents an important breakthrough in C programming tools. It sets you free so you can create exciting programs quickly and easily, saving both time and money! Developed to work with the IBM PC, XT, AT, compatibles, and MSDOS or PCDOS, DataWindows is a carefully tooled system of C functions which will jazz up your programs with unprecedented efficiency.

Greenleaf DataWindows is integrated windows, transaction data entry, pop-up, pull-down, and Lotus style menu systems with:

- **Screen Management.** You don't have to remember what's on the display or the sequence in which you put it there. DataWindows does the grunt work. There are no restrictions.
- **Transaction Data Entry.** Data entry windows can have any number of fields with sophisticated options for reading many data types. Calls are made to help, validation, and other functions. Full featured text editing, protected and mandatory fields, dBASE type picture strings, context sensitive help, validation of fields and transactions, redefinable keys, password entry, attribute control, keyboard idle and much more.
- **Device Independence.** It detects the type of display adapter your computer is using and adjusts to it automatically for CGA, EGA, or monochrome. Logical video attributes are easy to use for color or monochrome.
- **Compatibility.** Runs with Microsoft Windows and IBM TopView.
- **The Greenleaf Tradition of Quality.** Reliable products. Professional documentation that gets you up and running quickly and keeps you there. Reference card. Newsletter and Bulletin board.

IBM, Microsoft & dBase, are registered trademarks of International Business Machines, Microsoft Corporation & Ashton-Tate respectively. PCDOS, IBM PC, XT, AT, & TopView are trademarks of IBM; MSDOS and Microsoft Windows are trademarks of Microsoft Corporation.

Stop Window Shopping

Order Today. Or call toll free for a free demo of the windows library that makes all the others obsolete.

Order any of these high performance tools by calling your dealer or 1-800-523-9830 today. Specify compiler when ordering. Add $8 for UPS second day air, or $5 for ground. Texas residents add sales tax. MasterCard, VISA, P.O., check, COD. In stock, shipped next day.

Greenleaf DataWindows	$225
DataWindows Source Module	$225
The Greenleaf Comm Library v2.0	$185
The Greenleaf Functions v3.0	$185
Digiboard Comm/4-II	$325
Digiboard Comm/8-II	$535

GREENLEAF
Software©

1411 LeMay Drive, Suite 101
Carrollton, TX 75007

Call Toll Free
1-800-523-9830
In Texas and Alaska, call
214-446-8641

CIRCLE 97 ON READER SERVICE CARD

Window Dressings

- **Simple or Complex Windows.** Up to 254 powerful overlaid windows simultaneously, all with just one kind of window to remember! Yet any window can be from one character to 32K!
- **Easy Window Operations.** DataWindows lets you move, zoom, frame, title, change colors, titles, frames, size, location, and make windows visible or invisible at will! Functions set cursor, attributes, and write data to any window or "current window". Word wrap, auto scroll, keyboard functions.
- **Write to Any Window Any Time.** Windows may be visible, overlaid, or invisible, and you can write to them anyway. What you write will be seen when the windows become visible.
- **DataWindows is fast!** It writes directly to video memory (in some modes).
- **Easy to save!** Any window, complete with attributes, can be saved on disk quickly and efficiently.
- **Source code available. No royalties.**

Also from Greenleaf:

The Greenleaf Functions v3.0
The most complete, mature C language function library for the IBM PC, XT, AT and close compatibles. Includes over 225 functions — DOS, disk, video, color text and graphics, string, time/date, keyboard, disk status and Ctrl-Break functions plus many more.

The Greenleaf Comm Library
Our 2.0 version is the hottest communications facility of its kind. Over 120 functions — ring buffered, interrupt driven asynchronous communications for up to 16 ports simultaneously with XMODEM, XON/XOFF, many many sophisticated features.

We support all popular C compilers for MSDOS/PCDOS: Microsoft, Lattice, Computer Innovations, Aztec, DeSmet, and others.

COM PORT DRIVER
(continued from page 46)

stall excom after running the clock initialization. A bit of experimentation may be required. Excom is self-installing; just run excom to install it. If all you need is buffered input, you've finished. Excom will be properly set up by programs using the COM ports, and the existing mode command will configure excom. To tap the power of excom, you may need to access some of the extended features. If you use excom with your own programs, you can just make an *int 14h* call with *ah = 4* to set any extended options you may need. If you use excom with existing programs, you need a way to set the extended options from your keyboard or a batch file.

Listing Two, page 75, is a simple little C program called exmode that sets extended configuration parameters. Exmode works with the Microsoft 4.0 C compiler. If you have a different one, you may need to rewrite the function *int14*—it just sets registers and performs a software *int 14h*. You can also use exmode to install and remove excom, usually useful only while testing.

Exmode is self-documenting; just run exmode and it will tell you what it can do. Note that running exmode does not add to existing extended options but sets the complete set—that is, *exmode com1 nodsr* followed by *exmode com1 nocts* is not the same as *exmode com1 nodsr nocts*. In the first case, only *nocts* is left set; in the second, both *nocts* and *nodsr* are set. COM1's settings can be cleared by running *exmode com1*. The previous examples will not change any of COM2's settings.

An interesting possibility with excom is nesting several invocations of excom. Say you have excom set up the way you want it for a shell running on your CRT and want to temporarily run a communications program. Just install excom again, set it up for communications, run the communication program, and then use exmode to remove excom. The most recent copy of excom is removed, restoring your old excom untouched.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

Bibliography

Sargent, Murray; and Shoemaker, Richard L. *The IBM Personal Computer from the Inside Out*. Reading, Mass.: Addison-Wesley, 1984.

National Semiconductor Corp. *NSC800 Microprocessor Family Databook*. National Semiconductor, 1985.

International Business Machines Corp. *Technical Reference–Personal Computer XT and Portable Personal Computer*. IBM, 1984.

DDJ

(Listings begin on page 64.)

Vote for your favorite feature/article.
Circle Reader Service **No. 5**.

Dynamic Memory Overlays for Turbo Pascal

by Steve McMahon

A fast and versatile way to handle large programs in Turbo Pascal

Borland International's Turbo Pascal has a significant limitation that hinders its usefulness for developing large programs. Because the compiler can generate code for only a single code segment, it imposes a 64K limit on executable code space. Turbo Pascal's overlay system, designed to circumvent this restriction, potentially allows for much larger amounts of code but at the cost of dramatically slowing program execution as overlay code is loaded from disk files.

This article describes a way to use Turbo Pascal's overlay facility in conjunction with dynamic memory to build memory overlays that can be loaded and executed far more rapidly than can disk overlays. These memory overlays are easy to set up and can optionally be left on disk at run time if sufficient memory is lacking. This technique lets you have the best of both worlds: a program that can execute in limited memory with disk overlays but that can take advantage of extra memory to load overlays into RAM.

Turbo Pascal's Overlay Scheme

Borland's solution to the large-program problem is an overlay system. Programmers confronted with a program that would otherwise overrun the 64K code segment limitation can specify that the code of a group of procedures and functions should share the same execution space in the code segment. You make this declaration by grouping the routines contiguously in the source code and preceding the declaration of each overlay routine with the Turbo reserved word *overlay*. On compilation of the program, all the routines of an overlay group are compiled to a separate overlay file, where the executing program can find them as necessary. You can specify as many overlay groups as you wish within a program. The only coding limitation is that routines within an overlay group cannot call themselves (they cannot be directly recursive) or other routines within the same overlay group. (The recursion limitation can be circumvented by calling the overlay routine from a recursive, nonoverlay function or procedure.)

The advantages of this scheme are that it is easy to use and that it allows you to construct programs with code not only larger than 64K but also potentially larger than available RAM. The disadvantage is a significant performance degradation: disk activity and DOS file handling overhead are now required every time an overlay routine is executed. Careful selection and grouping of overlay routines can minimize such performance degradation, but it is still possible for a program to spend far more time reading overlays from disk than doing anything else.

Overlay performance can be significantly enhanced by storing the overlay file on a virtual memory, or RAM, disk. Turbo Pascal's *OvrPath* procedure makes this possible by allowing run-time specification of the location of overlay files. This solution to the performance problem, though, has requirements—that a RAM disk be available, overlay files be located there, and the program be correctly informed of that location—that are unacceptable for application programs intended for use in normal DOS environments by unsophisticated operators.

A solution to the overlay performance problem that is faster, more versatile, and completely invisible to an application program's users is presented here. But first, let's take a look at how Turbo Pascal's overlay system works.

How Overlays Work

When Turbo Pascal compiles a program that has overlay procedures, it reserves an overlay area in the code segment (and in the .COM file) large enough to accommodate the largest of the procedures or functions in a given overlay group. All the procedures and functions in that overlay set are then compiled for execution at the same position within the reserved overlay area. Those procedures and functions are stored all together in a separate overlay file that bears the same name as the main program but has the number of the overlay group (.000, .001, and so on) as a file-name extension.

When some other portion of the program makes a call to an over-

Steve McMahon, P.O. Box 3262, Berkeley, CA 94703. Steve has been working with Turbo Pascal since Version 1.0. His company, SunType Publishing Systems, develops software for the newspaper industry.

layed function or procedure, the calling code passes in registers the offset of the required overlay function or procedure inside the overlay file and the length of the code fragment to be loaded. The call is made to a fragment of code positioned at the top of the reserved overlay area. This fragment passes control to the overlay handler in Turbo Pascal's run-time package, having placed the address of the reserved overlay area on the stack. The overlay handler responds by checking to see if the code required is already in place in the overlay area (the overlay file offset of whatever code is currently in the overlay area is stored in a data area at the top of the reserved overlay space along with the name of the overlay file). If the code required is already in place, the overlay handler executes it. If not, the code is read from the overlay file: the overlay file is opened, a seek is made to the required offset, and the specified quantity of code is read into the overlay area. Finally, the file is closed, the overlay file offset of the now-current code fragment is saved in memory for future reference, and the overlay function or procedure is executed.

This overlay handling method is ripe for alteration. If the contents of the overlay file could be positioned in some otherwise unused portion of memory, then a replacement overlay handler could snatch code fragments from that area of memory rather than from a disk file. The time savings could be impressive: overlay calls would require no disk activity. Such a scheme should even provide significant speed advantages over using Borland's overlay handler in conjunction with a RAM disk (to contain the overlay file) because DOS file handling overhead would be eliminated. What a memory overlay scheme couldn't do, though, is allow for programs that are truly larger than memory because all the overlay code would need to fit in memory available to the program.

Such a scheme can be implemented with no changes to the compiler and with surprisingly little extra code. Best of all, it can be done with little modification of existing programs that use overlays. All that's necessary is to include the procedures contained in MemOvrly.Inc (Listing One, page 78) in a program that uses overlays and to add some initialization and deinitialization code to the program. (Some limitations on the scheme are discussed later.)

Using MemOvrly.Inc

You can include the three procedures that make up the memory overlay handler in an existing program by adding the line:

```
{$I MEMOVRLY.INC}
```

at a point in the procedure and function declaration part of a program that uses overlays (do not nest it inside a procedure or function). Then, you must add a procedure call, probably inside the main body of the program, to initialize the substitute overlay handler for each overlay group that you wish to use as a memory overlay group. You can add code to dispose of the dynamic memory used by the memory overlay group or groups at the end of the program. This cleanup code is particularly important if the program chains to another Turbo Pascal program because Turbo Pascal preserves the dynamic memory heap on chaining.

To set up a particular overlay group as a memory overlay group, all that's necessary is to run the *InitOverlay* procedure, passing it the address of some procedure or function in the overlay group. If, for example, an overlay group contained, among other routines, the function or procedure *One*, the entire overlay group

```
PROGRAM OverlayTest;

    (*  Memory Overlay Demonstration Program.  *)

    {$I MEMOVRLY.INC}

  VAR
    c : Char;

  OVERLAY PROCEDURE One;
    BEGIN
      WriteLn('This is Overlay Procedure One.');
    END;
  OVERLAY PROCEDURE Two;
    BEGIN
      WriteLn('This is Overlay Procedure Two.');
    END;

  BEGIN

    {Install the new overlay handler by passing it the address
    offset of ONE procedure or function from the overlay group.
    Multiple invocations for multiple overlay groups should be
    no problem.}

    InitOverlay(Ofs(One));

    REPEAT
      Write('Hit any key to run the overlays (^Z to stop): ');
      Read(Kbd, c);
      WriteLn;
      IF c <> ^Z THEN
        BEGIN
          One;
          Two;
        END;
      WriteLn;
    UNTIL c = ^Z;

    {Free up the heap space used by the replacement overlay
    handler by passing the same offset as above to the
    DisposeOverlayStorage Routine -- VITAL if you're chaining
    to another program.  The heap is preserved in a chain operation.}

    DisposeOverlayStorage(Ofs(One));

  END.
```

Example 1: *Short program demonstrating memory overlays*

TURBO PASCAL OVERLAYS
(continued from page 51)

could be initialized as a memory overlay group with the statement:

```
InitOverlay( Ofs( One ) );
```

Ofs is a built-in Turbo Pascal function that returns the offset within a segment of a variable or procedure. Because you know that the reserved overlay area associated with the routine *One* is in the code segment, passing the offset of the routine is sufficient to establish where the reserved overlay area for this group is located. This instruction allocates dynamic memory sufficient to contain all the overlay code, reads that code into memory, and installs the procedure *NewOverlayHandler* as overlay handler for that group. Any overlay groups not set up in this fashion are handled by Turbo Pascal's normal overlay handler. Also, if not enough dynamic storage is available to accept the overlay code, the group is left as a normal, nonmemory overlay.

Likewise, the memory used to contain all the overlay code for a given overlay group can be reclaimed for other uses with the instruction:

```
DisposeOverlayStorage( Ofs( One ) );
```

It's crucial that you do not try to call any routine in the overlay group after issuing this instruction. The *DisposeOverlayStorage* procedure does not restore the old, nonmemory overlay handler for the specified overlay group. It merely frees up the memory previously occupied by overlay code for other uses. So, the *DisposeOverlayStorage* command is usually used only (if at all) at the end of a program. If the program does not use Turbo Pascal's *Chain* facility to chain to another Turbo Pascal program, you probably don't even need to use the *DisposeOverlayStorage* procedure. If you do need it, it should be executed once for each memory overlay group in use.

How It Works

When asked to initialize a memory overlay group, the *InitOverlay* procedure pulls the name of the overlay code file from the reserved overlay area and opens the file as a Turbo Pascal untyped file. The file is sized and a check is made to make sure that enough dynamic memory is available to hold the entire file's contents with sufficient heap space left over to satisfy the program's other needs. (The amount of dynamic memory otherwise required by the program is determined by the constant *RequiredHeap*, which can be changed to reflect the needs of a particular program.)

If not enough dynamic memory is available, things go no further and the overlay group is left as a normal, nonmemory overlay group. Otherwise, heap space is allocated and the overlay file is read into memory. Then, the initialization procedure substitutes codes comprising a call instruction for the memory overlay handler for code that would normally call Turbo Pascal's native overlay handler. A data area that formerly contained the overlay file name is subsequently filled with information more pertinent to the memory overlay handling routine: the location and size of the heap space containing all the overlay code.

When a routine in the memory overlay group is needed, the call is directed to the new overlay handler. This handler uses the overlay offset and size location it receives to find the required code on the heap. The code is then moved into the reserved overlay area in the code segment with a string move and is executed.

The procedure that disposes of overlay storage on the heap is pretty straightforward. The only special thing it does is check to make sure that the overlay group it has been pointed to is actually a memory overlay group. If there wasn't adequate dynamic memory when memory overlay initialization was attempted, then the group might have been left as a normal overlay group and there would be no dynamic memory to free.

Limitations and Cautions

The memory overlay scheme described here does not work with overlay groups for which the overlay file is 64K or larger.

Nested overlays, created by declaring overlay groups within already overlayed functions or procedures, cannot be made into memory overlays. This would require modification of code inside overlay files or on the heap and seems beyond the bounds of what could be reliably, simply implemented.

Turbo Pascal's *OvrPath* procedure won't work in conjunction with memory overlay procedures. This is one limitation that could be overcome without too much difficulty by anyone who understands the code in MemOvrly.Inc—but it's probably not necessary because the primary utility of *OvrPath* is the placement of normal overlays on RAM disks.

MemOvrly.Inc is highly version dependent. It definitely won't work with any version of Turbo Pascal prior to 3.0, and if Borland changes its overlay handling technique in later versions, MemOvrly.Inc may have to be modified to take account of the changes.

Finally, as with overlays in general, memory overlays should be used only for functions and procedures that have been thoroughly debugged. Debugging functions and procedures placed in overlay groups can be exceptionally difficult.

Customizing MemOvrly.Inc

You may wish to customize the memory overlay handling routines in two ways. First, the initialization routine contains no input/output error checking code.

Second, when a program needs significant amounts of dynamic memory, the *RequiredHeap* constant at the top of MemOvrly.Inc should be customized to reflect that fact (as should the "minimum free dynamic memory" setting on the Turbo Pascal memory usage menu).

Availability

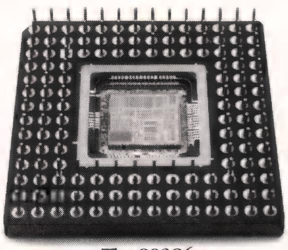
All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listing begins on page 78.)

Vote for your favorite feature/article.
Circle Reader Service **No. 6**.

A TRAINING COURSE FOR PEOPLE WHO LUST AFTER POWER.



The 80386

Intel is offering three new courses on the world's most powerful 32-bit microprocessor–the 80386. Plus a new intensive course on the 80286.

These in-depth learning sessions are designed for engineers and programmers who want to utilize the full power and potential of these lightning-fast chips.

Lectures are combined with hands-on workshops to provide real-life situations. Allowing you to apply new concepts and techniques immediately.

Courses include:

80386 System Software
80386 Programming using ASM386
High-end Microprocessor Hardware Design
The new 80286 Microprocessor Family Course

Complete training sessions and courses can be scheduled at your facility, or at our training centers.

In addition to training, Intel offers hardware/software support and consultants.

For more complete course information and schedules, call toll-free (800) 548-4725.

Or to register now, contact one of the Intel Training Centers listed below.

Intel Training Centers

Boston Area, Westford Corp. Ctr.
Three Carlisle Road, 1st Floor
Westford, MA 01886
(617) 692-1000

Chicago Area
300 N. Martingale Road, Suite 300
Schaumburg, IL 60194
(312) 310-5700

San Francisco Area
2700 San Tomas Expressway
Santa Clara, CA 95051
(408) 970-1700

Washington, D.C. Area
7833 Walker Drive, 5th Floor
Greenbelt, MD 20770
(301) 220-3380

intel®

A Unix BBS Using Shell Scripts

by Jan L. Harrington

> ***Unix has made it remarkably simple to develop a bulletin-board system.***

Two years ago I assumed the responsibility of developing a BBS (now known as Scholastech Telecommunications) for Scholastech, a nonprofit organization made up of college professors and other industry professionals whose goal is to facilitate and enhance academic computing. The BBS is designed to provide a distribution mechanism for Scholastech's public-domain/shareware software collection and to futher the exchange of information between educators at the college and university level.

Scholastech Telecommunications' software was originally developed and installed on an AT&T 7300 with 1-megabyte RAM and a 20-megabyte hard disk (by the time this article is published, the system will have been transfered to an AT&T 3B2 with a 72-megabyte drive). It has been in operation since October 1985, and to the best of AT&T's knowledge, it is the only BBS that has ever run on a 7300. Though the decision to use that particular machine was dictated only by circumstances (no other Scholastech hardware had the necessary configuration), the Unix operating system has provided an ideal environment in which to develop and operate a bulletin-board system. Not only is Unix multiuser and multitasking but it also contains significant support for telecommunications.

This article briefly discusses the Unix telecommunications environment and then looks in depth at the software written to implement Scholastech Telecommunications. Although it may seem like exposing the trade secret of the century, it is nonetheless true that the Unix operating system has made the development remarkably simple; the entire BBS software consists of a set of Unix V Bourne shell scripts, some supporting text files, and a public-domain XMODEM file transfer program.

Jan L. Harrington, 4002 Stearns Hill Rd., Waltham, MA 02154. Jan is an assistant professor in the Computer Science Department at Bentley College. She is the author of Macintosh Assembly Language: An Introduction.

The Unix Telecommunications Environment

Most implementations of Unix support both electronic mail (*mail*) and intersystem file transfers (*uucp*—Unix to Unix copy). When these functions are present, the system is able to answer and place phone calls. Unix also supports *cu* (call up) operations, which turn the system issuing the call into a dumb terminal. If the system called happens to be another Unix machine, *cu* supports file transfer as well. In terms of developing a BBS, *cu* is of little use because it is unreasonable to assume that all incoming calls will be from another Unix system. *Cu*'s file transfer functions therefore cannot be used as part of BBS software.

On the other hand, the presence of facilities to manage electronic mail and *uucp* transfers means that the system can handle incoming calls from any type of system (it is nontheless true that the full power of Unix telecommunications can only be realized when the communication is with another Unix system). The steps for preparing a Unix system for telecommunications do not vary greatly from one implementation to another, though the details of how and where the activities occur may be different. Generally, setting up a Unix system to receive incoming calls involves the following two actions:

1. Configure communications port(s): The AT&T 7300 has two internal modems; port configurations are handled by an applications shell known as The Office (because The Office performs some undocumented actions when it configures ports, users of the 7300 are wise not to attempt to do the configuration from the Bourne shell). More commonly, however, the configuration is handled from the shell by making appropriate entries in device configuration files, though the actual names of the files will vary between implementations.
2. Verify that a *getty* process is available for the port(s): *Getty* is a Unix system process that polls devices for input; it is also the process that actually logs a user onto the system. Instructions as to how the system should respond to device activity are contained in the file inittab. For example, the following inittab is used by the 7300:

```
is:2:initdefault
rc:bootwait:/etc/rc > /dev/window
                    < /dev/w1 2>&1
vid:2:respawn:/etc/getty window
                            9600
:ph0:2:respawn:/etc/getty ph0 1200
ph1:2:respawn:/etc/getty ph1 1200
000:2:respawn:/etc/getty tty000 9600
```

The first line in inittab puts the system in multiuser mode when the system is started; the second invokes *rc*, the shell script that handles system boot. The remaining lines are for actual devices. The colon in front of *ph0* indicates that the first modem line is inactive and should not be polled; the line in use is *ph1*. The word *respawn*, seen in the four device lines, indicates that if the *getty* process is not active when the device is polled, it should be started.

For some implementations of Unix, the same port cannot be used for both incoming and outgoing calls; the *getty* process must be inactive for outgoing lines and active for incoming lines. Configuring such systems for outgoing calls means that a colon must be placed in inittab at the beginning of the entry for each port that will be used to dial out, as was done above for the inactive *ph0* line. The 7300, however, kills the *getty* on a port automatically whenever an outgoing call is placed, making it possible to use the same line for both incoming and outgoing traffic.

Once one or more ports have been configured (assuming a modem is attached) and a *getty* is polling those ports, Unix answers incoming calls and takes users through the log-in process without intervention from an application program. Unix also intercepts the system log-out command, Ctrl-D, and logs users off. It is important to realize that this is not enough to configure the system for outgoing calls; outgoing calls require at least an entry in the file L-devices which describes the type of modem in use. *Uucp* and *mail* also require entries in the file L.sys, which identifies the names and phone numbers of other Unix systems that the host can call. Additional configurations may be necessary under specific Unix implementations.

What does this mean for a BBS that is to run on a Unix machine? The BBS software does not have to manage telecommunications: it does not have to poll a serial port to detect incoming calls, it does not have to log users onto the system, and it does not have to log users off the system. A simple Unix BBS needs to be concerned only with the actual functions remote users will perform after they are actually logged onto the system.

Overview of the System

The Scholastech BBS software provides the following functions:

- file upload and download
- public message exchange
- private mail
- user help
- change of password

Each user is given a separate Unix account under what is known as the "restricted shell." The restricted shell prevents users from changing directories, from accessing directories not in their default *PATH* (set in a .profile, an executable shell script that is run automatically when a user logs in), and from using any commands not contained in their rbin directory. The rbin directory is also made part of the user's default path. The restricted shell therefore effectively locks BBS users into a small segment of the system, preventing them from even listing the contents of directories that are not part of the BBS.

> **The steps for preparing a Unix telecommunications system do not vary greatly.**

BBS commands are each implemented as a separate shell script. There are at least four advantages to this approach:

1. The short shell scripts are easier to debug than a long, single program, regardless of whether the program is a shell script or compiled C.
2. New commands can be tested and added at any time without disturbing BBS operation.
3. Individual shell scripts simplify the logging of users' activities and the monitoring of their activities while they are on the system.
4. Shell script programming doesn't require the nearly 10 megabytes of disk space needed by the C development system on the 7300.

There are, of course, some major disadvantages to working with separate shell scripts:

1. Each user must have his or her own account (as long as the number of users is small, this is not an enormous problem, but as the number of users rises, account management begins to consume significant amounts of time).
2. Because they are interpreted rather than compiled, shell scripts run much more slowly than would, for example, programs in compiled C.
3. The Bourne shell itself, although containing powerful flow of control statements, is weak in terms of arithmetic operations. Even simple addition must be prefaced with the command *expr* and enclosed in single quotes before the sum can be assigned to a variable.

Scholastech Telecommunications has two special accounts without passwords, new and info. The new account is designed only for on-line account requests; the info account displays information about Scholastech activities and can accept on-line sign-up for Scholastech workshops. Code for these functions is contained in their .profiles.

BBS Directory Structure

Scholastech Telecommunications files are maintained within the directories /u/bbs and /u/bbs/rbin and their subdirectories. The contents of the directory in /u/bbs are:

- rbin directory (contains all commands available to restricted shell users)
- MS-files directory (contains MS-DOS software)
- Unix-files directory (contains Unix software)
- Mac-files directory (contains Macintosh software)
- Uploads directory (destination for all uploaded software)
- new (log-in for on-line sign-up for new accounts)
- info (log-in for Scholastech information and on-line workshop sign-up)
- msg directory (contains support files for the public message

(continued from page 55)

subsystem)
- log.file (a text file that records BBS command usage)
- msg1 (a text file whose contents are displayed by a user's .profile)
- directories for each BBS user

The directory /u/bbs/rbin contains:

- *dwnld* (shell script for downloading software)
- *help* (shell script for displaying BBS instructions)
- *list* (shell script for listing the contents of a data library)
- *pmail* (shell script for sending and reading private mail)
- *readmsg* (shell script for reading public messages)
- *scan* (shell script for viewing the headers of public messages)
- *send* (shell script for sending public messages)
- xmodem (public-domain file transfer program; compiled C)
- supporting text files:
  a. MS.list (listing of MS-DOS data library)
  b. Mac.list (listing of Macintosh data library)
  c. Unix.list (listing of Unix data library)
  d. help.file (text file with BBS instructions)
  e. user.file (listing of system users)
- all Unix commands used by the BBS shell scripts (either copied or linked into this directory)

The Shell Scripts

BBS User's .profile

The environment for each BBS user is configured by a short .profile:

```
PATH=/u/bbs/rbin:/u/bbs/MS-
files:/u/bbs/Mac-files:/u/bbs/Unix-
                    files:/u/bbs/Uploads
export PATH
PS1='> '
cat /u/bbs/msg1
echo
```

The .profile establishes the rbin directory and the program library directories as the user's default *PATH* (although technically there is only one restricted shell, /bin/rsh, there can be many rbin directories within a single Unix file system, each of which has a different path name and contains a different set of commands). The default prompt is also changed from the standard Bourne shell *$* to a >. Finally, the .profile displays the contents of a short welcome message (stored in msg1) and then returns control to the operating system.

Users logging in for the first time or after a long absence usually use the *help* command to display a screen full of instructions. The *help* command shell script is only two lines long:

```
echo 'who am i ¦ cut -f1 -d" " ' 'date ¦
cut -c1-c16' "help" >>/u/bbs/
                              log.file
cat /u/bbs/rbin/help.file
```

The first line writes a record to the command use log file, /u/bbs/log .file; the second simply displays the

contents of a text file containing instructions.

Users who wish to change their passwords can issue the command *passwd*. This Unix command is not contained in a shell script but is made available to BBS users at their > prompt. *Passwd* prompts for the new password and then asks that it be entered again for verification. On the 7300, *passwd* also enforces format constraints—passwords must be two words separated by a special character (only passwords established by the superuser can violate those constraints).

The Program Libraries

Three commands support user interaction with the program libraries—*list*, which displays the contents of a program library; *dwnld*, which downloads a file; and *upld*, which uploads a file.

The command *list* (Listing One, page 80) uses the Unix command *more* to display the contents of a text file that contains a listing of files available in a specific program library (lines 6, 11, and 16). Although the more widely used *cat* command also displays the contents of a text file, *more* displays the text one screen at a time (users advance by pressing the space bar or Return key) and is therefore better suited for screen displays. The Unix command to display a file directory, *ls*, could also be used to show the contents of the program libraries directly; this would eliminate the need to maintain the separate text files. In terms of security, however, *ls* is a "dangerous" command. If *ls* is present in the rbin directory, users who know Unix can use it from the > prompt to view the contents of that directory. They can then be aware that potentially more dangerous commands such as *mv*, which moves a file from one name or place to another, are available under the restricted shell. The best situation is to avoid the use of dangerous commands entirely, but they are essential to more than one of the BBS shell scripts.

The listings for each of the three program libraries are kept in separate files. Users must specify which program library as an argument to the command. For example, *list MS* displays the contents of /u/bbs/rbin/MS.list. The shell script traps the argument at lines 2, 7, and 12. If no argument is present (line 17), the shell script prints an error message (line 18) and exits. *List*, like all the other BBS commands, also writes a record to the command use log file (lines 4, 9, and 14).

Dwnld (Listing Two, page 80) transfers files from the program libraries to a remote user. The command requires two arguments—a designator for the program library and the name of the file. For example, *dwnld MS PCFILE.ARC* initiates transfer of the file /u/bbs/MS-files/PCFILE.ARC. The first 12 lines of the *dwnld* script identify the program library. If no program library is included, the script prints an error message (line 11) and does not execute a transfer. Assuming that a valid program library has been entered, *dwnld* then verifies that a file name is present (line 13); the error message for a missing file name appears in lines 37 and 38. The presence of a file name on the command line, however, does not guarantee that the file exists. A check for a valid file occurs on line 15 (the error message for an invalid file name appears in lines 32–34). After verifying that the requested file exists, *dwnld* writes a log record (line 17) and then prompts the user for the file transfer method (line 18). Files can be transferred as ASCII text (lines 24–29); the script simply *cat*s the file to the user, who must have instructed his or her terminal emulator program to capture incoming data to disk.

Binary file transfers are performed using the XMODEM file transfer protocol (line 22). Xmodem, the program used to implement the transfers, is a stand-alone file upload and download program that can be executed from the Unix command level as well as from within a shell script. Two versions exist in the public domain, umodem and uc, both of which are available as C source code. Xmodem was obtained by downloading the umodem source code from an existing Unix BBS and then compiling it on the 7300 (this was done prior to loading the program libraries,

(continued from page 57)

when there was still enough space of the disk for the entire development system). The xmodem command line is straightforward: the argument *-sb* indicates that a file is to be *s*ent and that its format is *b*inary; the second argument is the path name of the file to be transmitted.

Upld (Listing Three, page 81) handles file transfers in precisely the opposite manner to *dwnld*. ASCII uploads *echo* incoming characters to a text file (lines 14–19); to upload binary files, xmodem is invoked with the arguments *-rb*, *r*eceive *b*inary (line 12). *Upld* also contains code to verify that a name for the file to be uploaded has been entered as an argument to the command (lines 1 and 26–27); verifies that the file name for the incoming file doesn't duplicate an existing file name in the Uploads directory (lines 1–5); and collects a description of the file (appended to the text file /u/bbs/Uploads/Doc-.file), which the sysop can use to quickly figure out the nature of uploaded files.

The Public Message Subsystem

Three shell scripts support the exchange of public messages—scan displays the headers of all current messages, *readmsg* displays individual messages by number, and *send* sends a public message.

Scan is a short script that does nothing more than display the contents of a special text file, .index, to which an entry is added whenever a message is sent:

```
echo 'who am i ¦ cut -f1 -d" " ' 'date ¦
cut -c1-c16' "scan" >>/u/bbs/
                               log.file
echo
echo
more /u/bbs/msg/.index
echo
```

Scan's purpose is to allow users to identify the numbers of messages that might be of interest.

Messages are displayed on the screen by the command *readmsg* (Listing Four, page 81). *Readmsg* uses two simple files—/u/bbs/.first, which contains the number of the first message available, and /u/bbs/.last, which contains the number of the last message available—to remind users who might not have *scan*ned the message index of valid message numbers (lines 1–9) to do so. The file names of messages stored in /u/bbs/msg are the same as their message numbers. For example, message number 64 has the file name /u/bbs/msg/64. Therefore, once the user enters a message number (line 12), *readmsg* can simply *cat* a file by that name (line 18). The script also traps nonexisting message numbers (lines 19–21).

The shell script *send* (Listing Five, page 82) stores messages for the public message system. The script increments the last message number (line 8) and then conducts a dialogue with the user to collect the message header (lines 11–25) and the body of the message (lines 27–34). In addition, the message header must be added to the top of the file /u/bbs/msg/.index (that is, *scan* displays message headers in descending order by message number). To maintain the descending order, the headers are first written to a temporary file (lines 35–33). Then, the existing .index file is concatenated onto the end of the temporary file (line 39). Finally, the temporary file is moved on top of the old .index, effectively erasing the existing file (line 40). Because public messages can be addressed to groups of people (for example, *ALL* or *All 3B2 Users*), *send* does not bother to verify that the addressee is a valid system user ID.

The Private Mail Subsystem

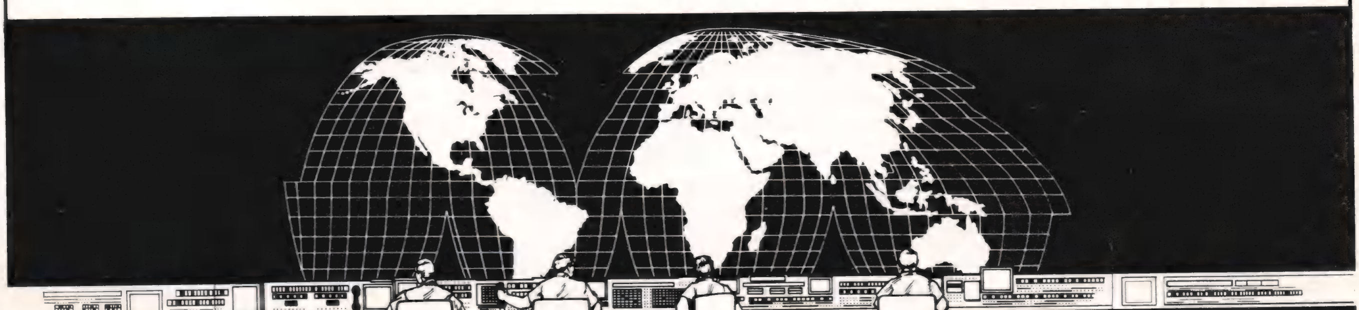
Private mail is built around the Unix *mail* command. The shell script *pmail* (Listing Six, page 82) builds an interface to aid users in sending and reading mail. The script first displays the four available commands (lines 4–11): *s* to send mail, *r* to read mail, *l* to see system users, and *x* to exit from *pmail*.

The *l* command is not a command normally associated with Unix mail, but sending Unix mail requires an argument of either a valid user ID or the electronic-mail name of a remote system known to the system from which the mail is sent. Unfortunately, user IDs are often unrelated to users' real names. It is therefore important for the BBS to provide some way for users to associate system user IDs with human names. *Pmail* displays the contents of the text file /u/bbs/rbin/user.file, which contains a listing of user names and user IDs (lines 61–63). The same information is also available from the system file /etc/passwd. In other words, it might be possible to avoid having to maintain user.file by using *cut* to obtain specific fields from /etc/passwd. The machine that supports Scholastech Telecommunications, however, also contains several accounts that are unrelated to the BBS. The code required to *grep* out the accounts that should not be displayed to BBS users far exceeds the effort to maintain *user.file*.

Sending Unix electronic mail is straightforward. The *mail* command is used with a single argument—the user ID of the recipient (although Unix mail can be directed to remote Unix systems, Scholastech Telecommunications does not make that capability available to BBS users). For example, *mail sysop* sends mail to the sysop account on the same system. To send private mail, the *pmail* shell script:

- displays a set of instructions that help ensure that the mail will be sent successfully (lines 19–29)
- collects the user ID of the account to which the mail is being sent (lines 31–32)
- verifies that the user ID is valid (lines 34–39)
- sends the mail by issuing the Unix *mail* command (lines 40–45)

Users working at the Unix command level can read mail by issuing the command *mail* without any arguments (there is no reason why BBS users who are knowledgeable about Unix cannot do so from their restricted shell > prompt). *Pmail* reads mail in exactly that way (line 55). Several options are available once a piece of mail has been displayed on the screen, including forwarding the mail to other users and saving the mail under a different file name in the user's account. *Pmail*, however, only alerts the user of the options of either deleting the message or leaving it in the user's mailbox (lines 51–54). Though knowledgeable Unix users may attempt to save the mail, BBS



(continued from page 58)

users do not have the right to write to their own accounts (this restriction was included to conserve disk space); any attempt to save mail in a user's account will therefore fail. Knowledgeable users may, however, succeed in forwarding mail to other BBS users.

Signing Up for a New Account

The special account new exists on Scholastech Telecommunications without a password. Its sole purpose is to permit prospective users to request accounts on-line. The code that manages the sign-up process is contained in the account's .profile (Listing Seven, page 86). Ideally, this .profile should automatically log users off the system once they have finished. The only way to do so that works on the 7300 (*stty 0* does not work) is to make the .profile the account's default shell. There is, however, an important reason why the account wasn't set up in that way; any shells other than /bin/sh (the standard Bourne shell) and /bin/rsh (the standard restricted shell) are assumed to be extremely restricted. In particular, they do not permit any redirection of output. That means that data cannot be sent to a disk file, something that is essential if user account request data are to be stored. Instead, the new account is established as a standard restricted shell account. Its default path is set to an rbin directory that contains only the two harmless commands *echo* and *cat* (line 1); its level 1 prompt is set to the phrase "Type CNTRL-D now . . ." (line 2). When the .profile ends or if a user breaks out of the .profile with the delete key, even those who know Unix will be powerless beause the commands in the rbin directory permit nothing but the display of text to the screen. The prompt continually reminds them of the key sequence to exit from the system.

Though somewhat lengthy for a .profile, the account request shell script is quite simple. It first verifies that the user wants to sign up for an account (line 5). Assuming the user does want an account, the script then enters a loop to collect name and address data (lines 13–43). After data have been entered, the user is given a chance to view the data (lines 33–38) and to correct it if necessary (lines 42–43). This process is repeated for the user ID and initial password (lines 47–60). Note that the system's password format is not enforced during the account request process. This is because initial passwords will be established by the superuser, bypassing the format restrictions. Finally, the script writes the data to a text file (lines 63–71).

The new account's .profile does not actually establish new accounts.

> ***I wrote two routines to keep records of system and command use.***

This must be done manually by completing the following steps:

1. Make an entry in /etc/passwd for the new user.
2. Enter a password for the new user.
3. Create a directory for the new user.
4. Copy the BBS .profile into the new user's directory.
5. Make an entry in /u/bbs/rbin/user.file for the new user.

Steps 1 and 2 must be performed by the superuser. The remaining steps are performed by the system operator (the account sysop). The sysop account owns all BBS accounts and their files. This prevents BBS users from viewing and/or modifying their own .profiles, an additional security measure often used along with the restricted shell.

Providing Organizational Information

The shell scripts that you have seen up to this point are generic; they might be used to support just about any BBS. The info account, however, provides a function needed by Scholastech that isn't typically found with most bulletin boards. Info's .profile (Listing Eight, page 88) works much like the new account's .profile to display information about upcoming Scholastech events and to allow users to register for workshops online.

Like new's .profile, the script first establishes a special rbin directory as the account's default path (line 1); info's .profile contains nothing but three text display commands—*cat*, *echo*, and *more*. It also changes the level 1 prompt to "Type CNTRL-D now . . ." (line 2). It then prints a welcome heading (lines 4–5) and displays the contents of a text file with the welcome message (line 7). The actual information about upcoming events is stored in the more extensive info .file, which is displayed page by page using the Unix command *more* (line 11).

Workshop sign-ups are straightforward. Users are prompted to enter registration data (lines 20–37). The .profile then appends that data to a text file (lines 38–43). The registration file is later printed and turned over to the Scholastech personnel in charge of the workshops.

System Monitoring

It is important for any BBS sysop to be aware of everything that is happening on the system. While users are logged in, their activites can be monitored with the Unix *ps* command, which displays the status of all active processes. There is, however, a further need to keep records of system and command use. Unfortunately, the 7300's implementation of Unix V does not include the standard Unix accounting functions. I therefore wrote two routines that display and reformat data from a system log file and from the log file written by each BBS shell script in order to obtain archival information.

The two shell scripts are *logs*, which displays data to the screen, and *usage*, which creates a printed report. Both rely on process beginning and ending data kept by Unix in the system file /etc/wtmp. Normally, /etc/wtmp is cleaned out every time the system is rebooted. This can, however, result in a loss of data whenever an unexpected power

Experience Prolog

PRICE BREAK BUY NOW

"ACTIVE PROLOG TUTOR (APT) is the best example of computer aided instruction I have seen."

Jim Loomis, Contributing Editor
Boston Computer Society

PROLOG CLEARLY & SIMPLY EXPLAINED

APT is designed to make learning easy, and most of all, fun. If you had the choice between reading text, looking at code, or experimenting on your own, what would you choose? Well, you don't have to choose with APT.

Prolog concepts are described in text form and then demonstrated by example using the Prolog interpreter. Use the interpreter as a practice environment to try your own ideas and get immediate feedback. A single keystroke toggles you between the tutorial text and the practice interpreter. Write a program and execute it without leaving the tutorial environment!

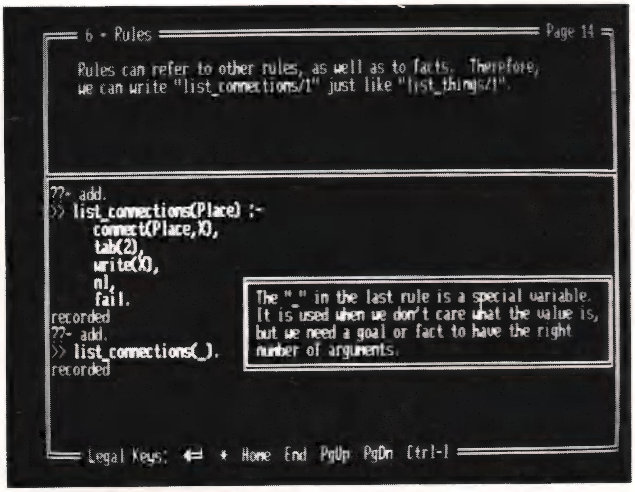
BUILD APPLICATIONS INTERACTIVELY

Learn by doing. The interpreter takes you step-by-step through the development of:

- an adventure game
- an intelligent genealogical database
- an order system
- an identification expert system
- a natural language user interface

As you progress, the programs get more sophisticated, building on the information you've already learned. By the end of the tutorial you have built many working Prolog applications.

The special Prolog interpreter included with APT is designed for education and is integrated with the tutorial package. It is not appropriate for production applications.



A typical APT screen will include a tutorial description of what is going on (top window), interpreter code (bottom window), and a pop-up window with helpful hints. Legal keys are always displayed on the bottom of the screen.

SYSTEM REQUIREMENTS

APT requires an IBM PC, AT or compatible hardware with 512K, and PCDOS 2.0.

SEEING IS BELIEVING

In order to understand the power of Prolog, it is important you know *how* it executes. Traditional teaching tools are ineffective in presenting what happens as Prolog executes. APT has 3 invaluable tracing mechanisms which will literally **show** you:

- the execution pattern of Prolog, and clause order importance,
- how values are passed up and down in a recursive Prolog routine, and
- how a Prolog query is processed.

EASY TO USE

- Manipulate the tutorial with single keystrokes.
- Move to prior pages easily.
- Instantly access any chapter from the table of contents.
- Automatically begin where you left off.
- All work is automatically saved when you leave the tutorial.
- The Prolog syntax you learn is compatible with:
  the Clocksin & Mellish standard, Arity Standard Prolog, and Arity Compiler/Interpreter

ORDER TODAY & SAVE

Order before June 30 and mention this ad to receive APT for $39. That's a 40% savings from the list price of $65.

CALL 800-821-2492

Solution Systems™

SOLUTION SYSTEMS, 335-F WASHINGTON ST., NORWELL, MA 02061. 617-659-1571

CIRCLE 148 ON READER SERVICE CARD

(continued from page 60)

failure occurs. I therefore made a modification to the system routine *.cleanup*, which is executed by the system boot routine *rc*, and turned the line > */etc/wtmp* into a comment to prevent its execution (I could have just as easily deleted it).

Logs (Listing Nine, page 90) first creates a temporary file (/u/user.log) that contains the user's name and the time logged on for each remote user with the Unix command *who*. A pipe to the Unix command *grep* extracts all entries that contain *ph1* (line 1). Because the device name for the internal modem being used by the BBS is *ph1*, the *grep* will include log-ins on that line and leave out all other devices (the console is usually either *w1* or *w2*, for example). If the *logs* command is issued with an argument *a* (line 2), *logs* displays the entire contents of both the temporary file (line 8) and the BBS command log file (line 15). If no argument is included, *logs* displays only the last ten lines of the two files (lines 21 and 26). Finally, *logs* removes the temporary file it created (line 29).

Usage (Listing Ten, page 90) creates two printed reports. The first is a columnar listing of the user's name, the date, the time on, and the time off for each log-in contained in /etc/wtmp. The second is simply a hard copy of the BBS command use file. *Usage* is a "dangerous" shell script; one of its functions is to reinitialize /etc/wtmp (line 17). It is therefore extremely important that no user other than the system operator should have any rights to the program (that is, its permission line should appear as *-rwx------*). Why shouldn't other users be allowed to read *usage*? Read rights allow another user to make of copy of a program to put in their own account. Once the copy is made, the user becomes the owner and can execute the script, wiping out /etc/wtmp and destroying system use data.

The logic behind *usage* is tied to the way in which data are stored in /etc/wtmp. There is one record for every process that starts and every process that ends (the two exceptions are the processes *LOGIN* and *rc*). The trick to capturing log-on and log-off times is therefore to match up the entries for any given user. Because the user running the program will not have logged off, the program must be careful to exclude that user from the report. The particular version of *usage* that appears in Listing Ten handles only remote users on the device *ph1*. In this case, the code doesn't need to worry about excluding the system operator who is running the program from the console. If, however, users on other devices are to be included, two things must happen:

It is extremely important that no user besides the sysop has rights to usage.

1. The code for assembling the report must be duplicated for each device (records for concurrent users are interleaved in /etc/wtmp).
2. The name of the user running the program must be excluded from the report because there will be no matching process termination record for that user.

The *usage* script first creates the system use report. It writes report headings to a temporary output file (lines 1–3) and then begins to assemble the body. The body is assembled in the following manner:

1. A log-on record for each user entering the system on *ph1* is extracted and stored in the file temp1 (line 4).
2. The first field of the record, the user name, is *cut* from temp1 and stored in temp2 (line 5). This forms the leftmost column of the report.
3. The log-on time is *cut* from temp1 and stored in temp3 (line 6). This forms the middle column of the report.
4. A log-off record for each user entering the system on *ph1* is extracted. The log-off time is *cut* from the record and stored in the file temp4 (line 7). This forms the rightmost column of the report.
5. The three temporary files containing the contents of the columns of the report are put together side by side with *paste* (line 8).
6. To complete the process, the report is formatted for output with the Unix *pr* command (line 9). The temporary files are removed (lines 10–14).

Records in the BBS command use log file (/u/bbs/log.file) are already in columnar format (they were written in such a manner by each BBS shell script). Therefore, they are simply formatted for output with *pr* (line 15). The log file is also reinitialized (line 16). Both reports are then sent to the printer (lines 18 and 19). Note that *usage* does not delete the two final output files because doing so before the files have actually been printed can disrupt the printing process.

Acknowledgment
I received both the XMODEM file transfer program and a great deal of help from David Watson, who operates a Unix/Xenix BBS in my area. Though I thanked him profusely at the time, I think he deserves a public acknowledgment for his generosity and patience in putting up with some of the elementary (dumb) questions I asked when I was first working on my BBS software.

Availability
All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listings begin on page 80.)

Listing One *(Text begins on page 42.)*

```
        title       extended int 14 handler
        name        excom
;
;       (C) 1987, Crystal Computer Consulting Inc.
;       This software may be used freely, at your own risk,
;       as long as this notice is not removed.
;
_text   segment  word public 'code'

        assume  cs:_text, ss:_text, ds:_text, es:_text

        org         0100h                           ; com-able

start:                                              ; must be first
        jmp         excom

        even
;       misc constants
BUFSZ   equ         0100h                           ; buffer size
XOFFSZ  equ         00F0h                           ; input flow control turn off point
XONSZ   equ         00E0h                           ; input flow control turn on point
DTR     equ         001h                            ; select DTR input flow control
RTS     equ         002h                            ; select RTS input flow control
XNXF    equ         004h                            ; select XON/XOFF input flow control
OUT2    equ         008h                            ; bit to set OUT2
NOCTS   equ         010h                            ; CTS not required to transmit
NODSR   equ         020h                            ; DSR not required to transmit
B19200  equ         040h                            ; set baud rate to 19200
B38400  equ         080h                            ; set baud rate to 38400
C1MASK  equ         010h                            ; com1 mask for 8259
C2MASK  equ         008h                            ; com2 mask for 8259

;       structure for port control blocks
pcbs    struc
putptr  dw          ?                               ; points to last char put
getptr  dw          ?                               ; points to next char to get
bufcnt  dw          ?                               ; number of characters in buffer
bufend  dw          ?                               ; address past buffer end
pbase   dw          ?                               ; port address
timeout db          ?                               ; timeout outer loop
mask    db          ?                               ; mask to enable port interrupt
rxoff   db          ?                               ; set if recv has been stopped
opts    db          ?                               ; holds extended options
lchar   db          ?                               ; holds character attributes
oldier  db          ?                               ; old interrupt enable register
oldlcr  db          ?                               ; old line control register
oldmcr  db          ?                               ; old modem control register
olddll  db          ?                               ; old baud low divisor latch
olddlm  db          ?                               ; old baud high divisor latch
buf     dw          BUFSZ dup (?)                   ; the buffer
pcbs    ends

;       allocate storage for port control blocks
pcb     pcbs        2 dup (<>)                      ; array of port control blocks

old0B   dd          ?                               ; old vector for int0B
old0C   dd          ?                               ; old vector for int0C
old14   dd          ?                               ; old vector for int14
oldmsk  db          ?                               ; old mask for 8259

;       baud rate table for UART initialization
bauds   dw          0417h                           ; divisor for 110 baud
        dw          0300h                           ; divisor for 150 baud
        dw          0180h                           ; divisor for 300 baud
        dw          00C0h                           ; divisor for 600 baud
        dw          0060h                           ; divisor for 1200 baud
        dw          0030h                           ; divisor for 2400 baud
        dw          0018h                           ; divisor for 4800 baud
        dw          000Ch                           ; divisor for 9600 baud
baud19  dw          0006h                           ; divisor for 19200 baud
baud38  dw          0003h                           ; divisor for 38400 baud

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                          ;
;   0Bh & 0Ch (com2 & com1) interrupt handler                              ;
;           put characters into the appropriate buffer                     ;
;           input flow control - stop input when buffer nears full         ;
;                                                                          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

int0B:
        ; point to com2 port control block with ds:si
        push        ds
        push        si
        mov         si, cs
        mov         ds, si
        lea         si, pcb + size pcbs
        jmp         short a10

int0C:
        ; point to com1 port control block with ds:si
        push        ds
        push        si
        mov         si, cs
        mov         ds, si
        lea         si, pcb

a10:
        push        di
        push        dx
        push        bx
        push        ax

        ; read character, if this fills buffer to XOFFSZ, then send
        ; XOFF or drop DTR and/or RTS as indicated by extended options
```

(continued on page 66)

Listing One *(Listing continued, text begins on page 42.)*

```
        mov     di, [si].putptr         ; buffer put pointer
        mov     dx, [si].pbase
        in      al, dx                  ; read character and clear interrupt
        mov     ah, al
        cmp     [si].bufcnt, XOFFSZ
        jl      a40                     ; jump if buffer below turnoff point
        cmp     [si].rxoff, 0
        jne     a30                     ; jump if receive already disabled
        mov     [si].rxoff, 1           ; indicate receiver now disabled
        test    [si].opts, XNXF
        jz      a20
        mov     al, 'S' - '@'
        out     dx, al                  ; send ^S
a20:
        mov     bl, [si].opts
        and     bl, DTR or RTS
        jz      a30
        add     dx, 4
        in      al, dx
        not     bl
        and     al, bl
        out     dx, al                  ; drop DTR and/or RTS
        sub     dx, 4
a30:
        ; if the buffer is full, set overflow status and exit
        cmp     [si].bufcnt, BUFSZ
        jl      a40
        or      word ptr [di], 0200h
        jmp     short a99
a40:
        ; read the port status
        add     dx, 5
        in      al, dx

        ; get new buffer pointer, adjust for wrap around if required
        inc     di
        inc     di
        cmp     di, [si].bufend
        jne     a50
        lea     di, [si].buf
a50:
        ; store new buffer pointer and put status:data into the buffer
        mov     [si].putptr, di
        xchg    ah, al
        mov     [di], ax
        inc     [si].bufcnt
a99:
        ; send interrupt complete command to interrupt controller
        mov     al, 020h
        out     020h, al
        pop     ax
        pop     bx
        pop     dx
        pop     di
        pop     si
        pop     ds
        iret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                            ;
;       14h interrupt handler                                                ;
;               emulate pc bios functions plus extensions                    ;
;                                                                            ;
;       dx is used to select the port for all calls                          ;
;               dx = 0 for com1,  dx = 1 for com2                            ;
;                                                                            ;
;       ah = 0  initialize the port, parameters in al                        ;
;               al bits 7-5 set the baud rate                                ;
;                 000 = 110,   001 = 150,   010 = 300,   011 = 600           ;
;                 100 = 1200,  101 = 2400,  110 = 4800,  111 = 9600          ;
;               al bits 4-3 set the parity                                   ;
;                 00 = none,  01 = odd,  11 = even                           ;
;               al bit 2 is number of stop bits                              ;
;                 0 = 1 stop bit,  1 = 2 stop bits                           ;
;               al bits 1-0 set the word length                              ;
;                 00 = 5,  01 = 6, 10 = 7, 11 = 8                            ;
;               returns ah & al as per comm status (ah=3) below              ;
;               example:  ah = 10101110 is 2400 baud,                        ;
;                         odd parity, 2 stop bits, 7 bit characters          ;
;                                                                            ;
;       ah = 1  transmit the character in al, al preserved                   ;
;               returns ah as per comm status (ah=3) below                   ;
;                                                                            ;
;       ah = 2  return receive character in al                               ;
;               returns ah as per comm status (ah=3) below                   ;
;               only bits 1-2-3-4-7 can be set                               ;
;               ah having any bits set is a receive error or timeout         ;
;                                                                            ;
;       ah = 3  return comm port status in ah & al                           ;
;               ah bits are:                                                 ;
;                 b0 = data ready          b1 = overrun                      ;
;                 b2 = parity error        b3 = framing error                ;
;                 b4 = break detect        b5 = xmit holding reg empty       ;
;                 b6 = xmit shift reg empty b7 = timeout                     ;
;               al bits are:                                                 ;
;                 b0 = delta CTS           b1 = delta DSR                    ;
;                 b2 = trail edge ring               b3 = delta recv detect  ;
;                 b4 = CTS                 b5 = DSR                          ;
;                 b6 = ring indicator                b7 = recv signal detect ;
;                                                                            ;
;       ah = 4  extended options                                             ;
;               returns 05A5A in ax, used to identify excom                  ;
;               al contains options as follows:                              ;
```

(continued on page 70)

...thanks!

When we founded Programmer's Connection in 1984, we dedicated ourselves to providing high quality personal service to every customer. Since then, we've quickly grown to be the leading independent dealer in this industry.

CELEBRATE WITH US

To celebrate our success, we're offering special sale prices on many of our products. We're also featuring some select, high-quality products on this page that represent exceptional values.

As we now enter our fourth year of business, we'd like to say "thanks!" to all of you who gave us the opportunity to serve you. And to those of you who haven't yet discovered our low prices and quality service, we extend a hearty invitation to give us a try. You'll be glad you did!

Turn the page for our latest advertised price list.

Sale prices effective through June 30, 1987.

Call or write for our FREE comprehensive price guide.

TURBOsmith by Visual Age is a powerful debugging environment for Turbo Pascal featuring **automatic,** multi-window readout of your source programs and variables. Trace and breakpoint with single-keystroke commands and watch the values of variables and expressions change. Supports Overlays, .COM and chain files. Synchronized Machine Code window has built-in assembler for in-line work.

List *$69* **Ours $59**

PERSONAL COBOL by Micro Focus is a fully-integrated program development system for creating sophisticated applications on your IBM Personal Computer or Personal System/2. This versatile package comes complete with a full-screen text editor, source code generator for screens, syntax checker, automatic file handling and an interactive source-level debugging facility.

List *$149* **Sale $99**

***PURCHASE A SUPPORTED C COMPILER** and receive 50% off any of the following libraries (when purchased on the same order):

	List	Without Language	*With Language
Data Management Consultants			
Zview Screen Mgmt Library	$245	$139	**$122.50**
Essential Software			
C Utility Library	185	119	**92.50**
Essential Graphics	250	183	**125.00**
Essential Comm Library	185	125	**92.50**
with Breakout Debugger	250	189	**125.00**

QUALITY. *We carry the finest selection of the best programmer's development tools specifically for IBM Personal Computers and compatibles. They are the latest versions and most come with return guarantees or evaluation periods.*

SUPPORT. *Our courteous, knowledgeable, non-commissioned salespeople are always ready to assist you. Our experienced technical consultants can answer questions about products and provide sound, unbiased advice.*

PRICE. *UPS Ground shipping is FREE to all U.S. customers. There are no extra charges for credit cards, CODs, purchase orders, sales tax (except Ohio) or special handling (except for export preparation). Quite simply, the prices on the next two pages are all you pay.*

INTEGRITY. *We're very proud of the trust we've earned from our customers and pledge always to be worthy of it.*

arity products

Arity Combination Package	1095	979
Expert System Development Pkg	295	229
File Interchange Toolkit	50	44
PROLOG Compiler & Interpreter	650	569
Screen Design Toolkit	50	44
SQL Development Package	295	229
Arity PROLOG Interpreter	295	229
Arity Standard Prolog	95	77

ai - expert systems

1st-CLASS *by Programs in Motion*	495	399
AutoIntelligence *by IntelligenceWare*	990	739
ExpertEDGE Advanced *by Human Edge*	2500	CALL
ExpertEDGE Professional *by Human Edge*	5000	CALL
Experteach II *by IntelligenceWare*	475	339
EXSYS *Development Software by EXSYS*	395	309
EXSYS Runtime System	600	469
Insight 1 *by Level Five Research*	95	75
Insight 2+ *by Level Five Research*	485	379
Intelligence/Compiler *IntelligenceWare*	990	739
Logic-Line Series 1 *by Thunderstone*	90	85
Logic-Line Series 2 *by Thunderstone*	125	115
Logic-Line Series 3 *by Thunderstone*	150	139

ai - lisp language

GCLISP *Golden Common LISP by Gold Hill*	495	CALL
GCLISP 286 Developer *by Gold Hill*	1190	CALL
IQCLISP *by Integral Quality*	300	CALL
IQLISP *by Integral Quality*	270	CALL
Microsoft LISP *Common LISP*	250	149
QNIAL *Combines LISP & APL by NIAL Systems*	375	339
Starter QNIAL *by NIAL Systems* *New*	99	89
TransLISP *from Solution Systems*	95	CALL
TransLISP PLUS *from Solution Systems*	195	CALL

ai - prolog language

APT *Active Prolog Tutor from Solution Systems*	65	CALL
LPA microPROLOG *All Varieties*	CALL	CALL
MPROLOG Language Primer *LOGICWARE*	50	45
MPROLOG P500 *by LOGICWARE*	495	395
MPROLOG P550 *by LOGICWARE*	220	175
Prolog-86 *from Solution Systems*	125	CALL
Prolog-86 Plus *from Solution Systems*	250	CALL
Turbo PROLOG *by Borland Intl*	100	63
Turbo PROLOG Toolbox *by Borland Intl*	100	64

ai - smalltalk language

Smalltalk/V *by Digitalk*	99	84
EGA/VGA Color Option	100	85
Goodies Diskette	49	45
Smalltalk/Comm	49	42

ai - texas instruments

PC Scheme *Lisp*	95	84
Personal Consultant Easy	495	435
Personal Consultant Plus	2950	2589
Personal Consultant Runtime	95	84

apl language

APL*PLUS/PC *by STSC*	595	424
APL*PLUS/PC Spreadsheet Mgr *by STSC*	195	139
APL*PLUS/PC Tools Vol 1 *by STSC*	295	199
APL*PLUS/PC Tools Vol 2 *by STSC*	85	58
ATLAS*GRAPHICS *by STSC* *New*	CALL	CALL
Financial/Statistical Library *by STSC*	275	189
Pocket APL *by STSC*	95	69
STATGRAPHICS *by STSC*	795	579

assembly language

386 ASM/LINK *Cross Asm by Phar Lap*	495	389
8088 Assembler *w/ Z-80 Translator by 2500 AD*	100	89
Advanced Norton Utilities *by Peter Norton* *New*	150	99
ASMLIB *Function Library by BC Assoc*	149	125
asmTREE *B-Tree Dev System by BC Assoc*	395	339
Cross Assemblers *Various by 2500 AD*	CALL	CALL
Microsoft Macro Assembler	150	93
Norton Utilities *by Peter Norton*	100	59
screenplay *by Flexus*	100	79
Turbo EDITASM *by Speedware*	99	84
Uniware Cross Assemblers *Various by SDS*	CALL	CALL
Visible Computer: 8088 *Software Masters*	80	64

basic language

87 Software Pak *by Hauppauge*	180	149
EXIM Services Toolkit *by EXIM*	50	45
Finally *by Komputerwerks*	99	85
Inside Track *from Micro Help*	65	51
MACH 2 *by Micro Help*	69	55
Microsoft QuickBASIC Compiler	99	63
Peeks 'n Pokes *from MicroHelp*	45	37
Professional BASIC *by Morgan*	99	68
8087 Math Support	50	42
QuickPak *by Crescent Software*	69	59
Scientific Subroutine Library *by Peerless*	125	99
Stay-Res *by MicroHelp*	95	73
True Basic *w/ Run-time*	245	179
True Basic	150	97
Run-time Module	150	97
Various Utilities	50	41
Turbo BASIC Compiler *by Borland Intl*	100	64

blaise products

ASYNCH MANAGER *Specify C or Pascal*	175	119
C TOOLS PLUS	175	119
EXEC *Program Chainer*	95	65
LIGHT TOOLS *for Datalight C*	100	CALL
PASCAL TOOLS	125	94
PASCAL TOOLS 2	100	CALL
PASCAL TOOLS & TOOLS 2	175	119
RUNOFF *Text Formatter*	50	39
TURBO ASYNCH PLUS	100	CALL
TURBO POWER TOOLS PLUS	100	CALL
VIEW MANAGER *Specify C or Pascal*	275	179

borland products

EUREKA *Equation Solver*	100	64
Reflex & Reflex Workshop	200	128
Reflex *Data Base System*	150	89
Reflex Workshop	70	45
Sidekick & Traveling Sidekick	125	85
Sidekick	85	57
Traveling Sidekick	70	45
Superkey *New Version*	100	64
Turbo BASIC Compiler	100	64
Turbo C Compiler *New*	100	64
Turbo Database Toolbox	70	41
Turbo Editor Toolbox	70	41
Turbo Gameworks Toolbox	70	41
Turbo Graphix Toolbox	70	41
Turbo Jumbo Pack *Combination Package* *New*	300	219
Turbo Lightning	100	64
Turbo PASCAL Numerical Methods Toolbox	100	64
Turbo PASCAL and Tutor	125	85
Turbo PASCAL	100	64
Turbo Tutor	40	24
Turbo PROLOG Compiler	100	63
Turbo PROLOG Toolbox	100	64
Word Wizard	70	47
Word Wizard and Turbo Lightning	150	94

c++

C++ *by Guidelines w/ version 1.1 kernel*	195	172
PforCe++ *Function Library by Phoenix*	395	225

c compilers

C86PLUS *by Computer Innovations*	497	CALL
Datalight C Compiler *Small Model*	60	43
Datalight Developer Kit	99	74
Datalight Optimum-C	139	99
DeSmet C *w/ Debugger & Large Case*	209	184
DeSmet C *w/ Debugger Only*	159	138
Eco-C *with Graphics by Ecosoft*	125	83
Lattice C Compiler *vers. 3.2 from Lattice* *Sale*	500	249
Mark Williams Let's C Combo Pack *New Version*	125	99
Let's C *Compiler* *New Version*	75	54
csd *Source Level Debugger* *New Version*	75	54
Microsoft C *with CodeView*	450	269
Turbo C Compiler *by Borland Intl* *New*	100	64
Uniware 68000/10/20 Cross Compiler *by SDS*	CALL	CALL

c interpreters

C-terp *by Gimpel, Specify compiler*	300	219
C Trainer *with Book by Catalytix*	122	87
Instant C *by Rational Systems* *Sale*	500	349
Introducing C *by Computer Innovations*	125	CALL
Run/C *by Age of Reason*	120	79
Run/C Professional *by Age of Reason*	250	157

c utilities

C to dBase *by Computer Innovations*	150	CALL
c-tree & r-tree Combo *by FairCom* *Sale*	650	449
c-tree *ISAM File Manager* *Sale*	395	269
r-tree *Report Generator* *Sale*	295	199
C Windows *by Syscom*	100	85
C Wings *by Syscom*	50	43
CI ROMPac *by Computer Innovations*	195	CALL
dBx *dBase to C Translator by Desktop AI*	350	299
with Library Source Code	550	469
Various Support Utilities	CALL	CALL
EASY SCREEN *by Retail Mgmt Systems* *New*	225	199
Entelekon Products	CALL	CALL
Flash-up Windows *by Software Bottling*	90	78
GraphiC *Color version by Sci Endeavors*	350	282
GRAFLIB *by Sutrasoft*	175	159
HALO *Graphics by Media Cybernetics*	300	205
HALO Development Pkg *for Microsoft* *Sale*	595	359
The HAMMER *by OES Systems*	195	129
PANEL *Forms Management by Roundhill*	295	CALL
PANEL Plus *by Roundhill* *New*	495	CALL
PC Lint *by Gimpel Software*	139	99
PLOTHI *by Sutrasoft*	175	159
PLOTHP *by Sutrasoft*	175	159
Professional C Windows *by Washburn*	89	79
Scientific Subroutine Library *by Peerless*	175	128
screenplay *for C by Flexus*	CALL	CALL
Vitamin C *by Creative Programming*	225	158
VC Screen *Forms Designer*	100	79
Zview *by Data Mgmt Consultants* *See First Page*	245	139

cobol language

COBOLspII *by Flexus* *New*	CALL	CALL
FPLIB *for Realia COBOL by BC Associates* *New*	149	129
Micro Focus COBOL *See Micro Focus Section*		
Microsoft COBOL *See Microsoft Section*		
Realia COBOL with RealMENU *New*	1145	899
Realia COBOL	995	783
RealCICS	995	783
RM/COBOL *by Ryan-McFarland*	950	CALL
RM/COBOL 85 *by Ryan-McFarland*	1250	CALL
screenplay *for COBOL by Flexus*	175	129

debuggers & profilers

386 DEBUG *Cross Debugger by Phar Lap*	195	129
Advanced Trace-86 *by Morgan Computing*	175	115
CI Probe *by Computer Innovations*	225	CALL
Codesifter *Profiler by David Smith*	119	85
Codesmith-86 *by Visual Age*	145	98
DSD87 *by Soft Advances*	125	99
MiniProbe *by Atron*	395	CALL
Periscope I *with Board by Periscope*	345	289
Periscope II *with NMI Breakout Switch*	175	139
Periscope II-X *Software only*	145	105
Periscope III *w/Advanced Board* *New*	995	CALL
The PROFILER *with Source Code by DWB*	125	89
TURBOsmith *for Turbo Pascal* *See First Page*	69	59
The WATCHER *Profiler by Stony Brook*	60	51

dos utilities

Command Plus *by ESP Software*	80	69
FANSI-CONSOLE *by Hersey Micro*	75	62
Norton Commander *by Peter Norton*	75	55
Scroll & Recall *by Opt-Tech Data*	69	59
Taskview *by Sunny Hill Software*	80	55

essential products

See First Page for Special Offers.

C Essentials *by Essential Software*	100	75
C Utility Library	185	119
Essential Comm Library *with Debugger*	250	189
Essential Comm Library *Software Only*	185	125
Breakout Debugger Only *Any language*	125	89
Essential Graphics	250	183

forth language

CFORTH *Native Code Compiler by LMI*	300	229
Forth/83 Metacompiler *Specify Target*	750	599
PC/Forth *by Laboratory Microsystems*	150	109
PC/Forth+ *by Laboratory Microsystems*	250	199
Advanced Color Graphics Support	100	74
Enhanced Graphics Support	200	148
Intel 8087 Support	100	74
Interactive Symbolic Debugger	100	74
Native Code Optimizer	200	148
Software Floating Point	100	74
UR/Forth *by LMI*	350	279
Object Module Libraries	500	395

fortran language

50 MORE: FORTRAN *by Peerless Engr*	125	95
ACS Time Series *Alpha Computer Service*	495	389
Essential Graphics *by Essential Software*	250	183
For-Winds *Alpha Computer Service*	90	69
Forlib-Plus *Alpha Computer Service*	70	44
FORTLIB *by Sutrasoft*	125	109
FORTRAN Addendum *by Impulse Engr*	95	85
FORTRAN Addenda *by Impulse Engr*	165	138
GRAFLIB *by Sutrasoft*	175	159
HALO *by Media Cybernetics*	300	205
I/O PRO *by MEF Environmental*	149	129
Microcompatibles Combo Package	240	215
Grafmatic	135	117
Plotmatic	135	117
Microsoft FORTRAN *w/ CodeView*	450	269
No Limit *by MEF Environmental*	129	109
Numerical Analyst *by MAGUS*	295	CALL
PANEL *by Roundhill Computer Systems*	295	CALL
PLOTHI *by Sutrasoft*	175	159
PLOTHP *by Sutrasoft*	175	159
RM/FORTRAN *Ryan-McFarland*	595	CALL
RTC PLUS *Fortran to C by Cobalt Blue*	325	CALL
Scientific Subroutine Lib *by Peerless*	175	128
Statistician *Alpha Computer Service*	295	235
Statlib.GL: *by PSISystems*	295	239
Statlib.TSF: *by PSISystems*	295	239
Strings & Things *Alpha Computer Service*	70	45

greenleaf products

Greenleaf Comm Library	185	125
Greenleaf Data Windows	225	157
with Source Code	450	289
Greenleaf Functions	185	125

help utilities

HELP/Control *by MDS*	125	99
On-line Help *from Opt-Tech*	149	99
SoftScreen/HELP *by Dialectic Systems*	195	149

lattice products

Lattice C Compiler *ver 3.2 from Lattice* *Sale*	500	249
with Library Source Code	900	495
C Cross Reference Generator	50	37
with Source Code	200	139
C-Food Smorgasbord *Function Library*	150	95
with Source Code	300	179
C-Sprite *Source Level Debugger*	175	119
Curses *Screen Manager*	125	85
with Source Code	250	169
dBC *Specify dBC II or dBC III*	250	169
with Source Code	500	356
dBC III Plus	750	CALL
with Source Code	1500	CALL
LMK *Make Facility*	195	138
RPG II Combo *All three items below*	1100	875
RPG II Compiler *No Royalties*	750	625
RPG II SEU *Screen Entry Utility*	250	199
RPG II Sort/Merge	250	199
RPG II Screen Design Aid Utility	350	309
SecretDisk *File Encryption Utility*	120	88
SideTalk *Resident Communications*	120	88
SSP/PC *Scientific Subroutine Library* *Sale*	350	229
Text Management Utilities	120	88
TopView Toolbasket *Function Library*	250	178
with Source Code	500	356

metagraphics products

LightWINDOWS/C *for Datalight C* *New*	95	79
MetaWINDOWS *No Royalties*	185	109
MetaFONTS	80	58
MetaWINDOWS/Plus	235	184
MetaFONTS/Plus	235	184
TurboWINDOW *Graphics/Windows for Turbo Pascal*	80	58

micro focus products

Micro Focus Level II COBOL *w/Animator* *New*	495	395
Level II COBOL	349	279
Level II Animator	195	155
Micro Focus Level II COBOL/ET for UNIX *New*	CALL	CALL
Micro Focus Personal COBOL *See First Page*	149	99
Micro Focus Professional COBOL	2000	1595
Micro Focus VS COBOL/XENIX *New*	1495	1195

Support Products:		
COBOL/IQ *Ad hoc Report Writer* . . . . New	495	395
COBOL/IQ *for DOS 3.X Networks* . . . . New	995	795
FORMS-2	295	235
SOURCEWRITER	995	795

microport products

System V/AT *by Microport Systems* . . . . Sale	549	459
Runtime System *(Operating System)*	199	189
Software Development System	249	235
Text Preparation System	199	189
User Upgrade *3 to Unlimited Users*	249	235

microsoft products

Microsoft BASIC Interpreter *for XENIX*	350	209
Microsoft C *with CodeView*	450	269
Microsoft COBOL *w/ COBOL Tools* . . . . New Version	700	429
for XENIX	995	609
Microsoft FORTRAN *w/ CodeView*	450	269
for XENIX	695	419
Microsoft Learning DOS	50	36
Microsoft LISP *Common LISP*	250	149
Microsoft MACH 10 *w/ Mouse & Windows*	549	369
Microsoft MACH 10 *Board only*	399	279
Microsoft Macro Assembler	150	93
Microsoft Mouse *Bus Version*	175	114
Microsoft Mouse *Serial Version*	195	124
Microsoft muMath *Includes muSIMP*	300	179
Microsoft Pascal Compiler	300	179
for XENIX	695	419
Microsoft QuickBASIC Compiler	99	63
Microsoft Sort	195	125
Microsoft Windows	99	63
Microsoft Windows Development Kit	500	299

modula-2 language

IOTools *by Rhodes Associates*	80	69
with Source Code	950	CALL
MODULA-2 Apprentice Pkg by LOGITECH	99	79
MODULA-2 Magic Pkg by LOGITECH	99	79
MODULA-2 ROM Pkg & Cross RT Debugger	299	239
MODULA-2 Window Pkg by LOGITECH	49	39
MODULA-2 Wizard's Pkg by LOGITECH	199	159
REPERTOIRE *for MODULA-2 by PMI*	89	79
Object Code Only	19	15

mouse products

LOGIMOUSE BUS *with PLUS Pkg by LOGITECH*	119	98
with PLUS & PC Paintbrush	149	119
with PLUS & CAD Software	189	153
with PLUS & CAD & Paint	219	179
LOGIMOUSE C7 *with PLUS Pkg, Specify Connector*	119	98
with PLUS & PC Paintbrush	149	119
with PLUS & CAD Software	189	153
with PLUS & CAD & Paint	219	179

other languages

CCS MUMPS *Single-User by MGlobal*	60	50
CCS MUMPS *Single-UserMulti-Tasking*	150	129
CCS MUMPS *Multi-User*	450	359
Janus/ADA C Pak *by R&R Software*	95	84
Janus/ADA D Pak *by R&R Software*	900	769
Janus/ADA ED Pak *by R&R Software*	395	CALL
Marshal Pascal *by Marshal Language Systems*	189	155
Personal REXX *by Mansfield Software*	125	99
SNOBOL4+ *by Catspaw*	95	80

other products

Dan Bricklin's Demo Pgm *Software Garden*	75	57
Dan Bricklin's Demo Tutorial . . . . New	50	45
Disk Optimizer *by Softlogic Systems*	60	55
FASTBACK *by 5th Generation Systems*	179	133
Informix *All Varieties by Informix*	CALL	CALL
Instant Replay *by Nostradamus* . . . . New Version	CALL	CALL
MKS Toolkit *with vi Editor by MKS*	139	99
Net-Tools *by BC Associates*	149	129
OPT-Tech Sort *by Opt-Tech Data Proc*	149	99
PC/TOOLS *by Custom Software* . . . . New	49	45
Screen Machine *by MicroHelp* . . . . New	79	59
VTEK *Term Emulator by Sci Endeavors*	150	129

phoenix products

Pasm86 *Macro Assembler Version 2.0*	195	108
Pdisk *Hard Disk & Backup Utility*	145	99
Pfantasy Pac *Phoenix Combo*	995	619
Pfinish *Execution Profiler*	395	225
Pfix86plus *Symbolic Debugger*	395	225
PforCe *Comprehensive C Library*	395	225
PforCe++ *Library for Guidelines C++*	395	225
Plink86plus *Overlay Linker*	495	299
Pmaker *Make Utility*	125	78
Pmate *Macro Text Editor*	195	108
Pre-C *Lint Utility*	295	154
Ptel *Binary File Transfer Program*	195	108

polytron products

PolyBoost *The Software Accelerator*	80	64
PolyDesk III . . . . New	99	72
PolyDesk III Archivist . . . . New	50	42
PolyDesk III Cryptographer . . . . New	50	42
PolyDesk III Talk . . . . New	70	52
PolyLibrarian *Library Manager*	99	73
PolyLibrarian II *Library Manager*	149	109
PolyMake *UNIX-like Make Facility*	149	109
PolyShell	149	109
Polytron C Beautifier	50	42
Polytron C Library I	99	72
Polytron PowerCom *Communications*	139	105
PolyXREF *Complete Cross Ref Utility*	219	169
PolyXREF *One language only*	129	99
PVCS Corporate *Version Control System*	395	309
PVCS Personal	149	109
PVMFM *Polytron Virtual Memory File Mgr* . . . . New	CALL	CALL

program mgmt utilities

Compact Source Print *by Aldebaran*	55	44
Interactive EASYFLOW *by Haventree*	150	125
PrintQ *by Software Directions*	89	84
Quilt Computing Combo Package . . . . New Version	250	199
QMake *Program Rebuild Utility*	99	79
SRMS *Software Revision Mgmt Sys* . . New Version	185	159
Source Print *by Aldebaran Labs*	75	59
TLIB *by Burton Systems Software*	100	89
Tree Diagrammer *by Aldebaran Labs*	55	48

raima products

dbQUERY *Single-User Query Utility*	195	129
Single-User with Source Code . . . . Sale	495	329
Multi-User . . . . Sale	495	329
Multi-User with Source Code . . . . Sale	990	659
dbVISTA *Single-User DBMS*	195	129
Single-User with Source Code . . . . Sale	495	329
Multi-User . . . . Sale	495	329
Multi-User with Source Code . . . . Sale	990	659

sco products

Complete XENIX System V *by SCO*	1295	994
Development System	595	499
Operating System *Specify XT or AT*	595	499
Text Processing Package	195	144
Networks for XENIX *by SCO*	595	495
SCO Professional *Lotus clone for XENIX*	795	595

softcraft products

Btrieve *ISAM Mgr with No Royalties*	245	184
Xtrieve *Query Utility*	245	184
Report Option *for Xtrieve*	145	99
Btrieve/N *for Networks*	595	454
Xtrieve/N	595	454
Report Option/N *for XtrieveN*	345	269

text editors

Brief & dBrief Combo *from Solution Systems*	250	CALL
Brief *Programmer's Text Editor*	195	CALL
dBrief *Customizes Brief for dBase III*	95	CALL
Epsilon *Emacs-like editor by Lugaru*	195	147
KEDIT *by Mansfield Software*	125	98
Micro/SPF *by Phaser Systems*	175	139
PC/VI *by Custom Software Systems*	149	99
SPF/PC *by Command Technology Corp* . . . New Version	245	CALL
Vedit *by CompuView*	150	98
Vedit Plus *by CompuView*	185	128

turbo pascal utilities

ALICE *Interpreter by Software Channels*	95	66
DOS/BIOS & Mouse Tools *by Quinn-Curtis*	75	67
Flash-up Windows *by Software Bottling*	90	78
MACH 2 *for Turbo Pascal by Micro Help* . . . . New	69	55
MetraByte D/A Tools *by Quinn-Curtis*	100	89
Science & Engrg Tools *by Quinn-Curtis*	75	67
Screen Sculptor *by Software Bottling*	125	91
screenplay *for Turbo Pascal by Flexus*	100	79
Speed Screen *by Software Bottling*	35	32
System Builder *by Royal American*	100	CALL
IMPEX *Query Utility*	75	CALL
Report Builder	75	CALL
TDebugPLUS *by TurboPower Software*	60	49
Turbo EXTENDER *by TurboPower Software*	85	64
Turbo Professional *by Sunny Hill*	70	45
TurboHALO *from IMSI*	129	98
TurboPower Utilities *by TurboPower*	95	78
TurboRef *by Gracon Services*	50	45
TURBOsmith *Visual Age Debugger* . . . . See First Page	69	59
Universal Graphics Library *by Quinn-Curtis* . . . New	150	119

wendin products

Operating System Toolbox . . . . Rebate Offer	99	75
PCNX *Operating system* . . . . Rebate Offer	99	75
PCVMS *Similar to VAXVMS* . . . . Rebate Offer	99	75
XTC *Text Editor w/ Pascal source* . . . . Rebate Offer	99	75

xenix/unix products

Btrieve *ISAM File Mgr by SoftCraft*	595	454
C-terp *by Gimpel, Specify compiler*	498	379
c-tree *ISAM Mgr by FairCom* . . . . Sale	395	269
dbVISTA *See Raima Section*		
dBx *with Library Source by Desktop AI*	550	489
DOSIX *Console Version by Data Basics*	399	CALL
DOSIX *User Version by Data Basics*	199	CALL
Informix *All Varieties by Informix*	CALL	CALL
Lyrix *by Informix*	595	449
Micro Focus Level II Compact COBOL	1000	CALL
Forms-2	400	CALL
Level II ANIMATOR	600	CALL
Microport Products *See Microport Section*		
Microsoft Products *See Microsoft Section*		
PANEL Plus *by Roundhill Computer Systems* . . . . New	CALL	CALL
REAL-TOOLS *Binary Version by PCT*	149	89
Library Source Version	399	289
Complete Source Version	499	369
RM/COBOL *by Ryan-McFarland*	1250	CALL
RM/FORTRAN *by Ryan-McFarland*	750	CALL
SCO Products *See SCO Section*		

LOWEST PRICES
Due to printing lead times, some of our current prices may differ from those shown here. Call for latest pricing.

FREE SHIPPING
Orders within the USA (including Alaska & Hawaii) are shipped FREE via UPS. Express shipping is available at the shipping carrier's standard rate with no rush fees or handling charges. To avoid delays when ordering by mail, please call first to determine the exact cost of express shipping.

CREDIT CARDS
VISA and MasterCard are accepted at no extra cost. Your card is charged when your order is shipped. Mail orders please include credit card expiration date and telephone number.

CODs AND POs
CODs and Purchase Orders are accepted at no extra cost. POs with net 30-day terms are available to qualified US accounts only.

FOREIGN ORDERS
Shipping charges for foreign and Canadian orders are based on the shipping carrier's standard rate. Since rates vary between carriers, please call or write for the exact cost. Foreign orders (except Canada), please include an additional $10 for customs form preparation. All payments must be made with US funds drawn on a US bank. Please include your telephone number when ordering by mail. Due to government regulations, we cannot ship to all countries.

VOLUME ORDERS
Volume orders may qualify for additional discounts. Call us for special pricing.

SOUND ADVICE
Our knowledgeable technical staff can answer technical questions, assist in comparing products and send you detailed product information tailored to your needs.

30-DAY GUARANTEE
Most of our products (excluding books) come with a 30-day documentation evaluation period or a 30-day return guarantee. Please note that some manufacturers restrict us from offering guarantees on their products. Call for more information.

MAIL ORDERS
Please include your telephone number on all mail orders. Be sure to specify computer, operating system and any applicable compiler or hardware interface(s). Send mail orders to:

Programmer's Connection
136 Sunnyside Street
Hartville, OH 44632

CALL TOLL FREE

U S	**800-336-1166**
CANADA	**800-225-1166**
OHIO & ALASKA (Call Collect)	**216-877-3781**
TELEX	**9102406879**
FOREIGN	**216-877-3781**
CUSTOMER SERVICE	**216-877-1110**

Hours: Weekdays 8:30 AM to 8:00 PM EST.

Ohio customers add 6% state sales tax.
Prices are subject to change without notice.

programmer's connection

COM PORT DRIVER

Listing One *(Listing continued, text begins on page 42.)*

```
;                       bit 0 enables DTR input flow control               ;
;                       bit 1 enables RTS input flow control               ;
;                       bit 2 enables XON/XOFF input flow control          ;
;                       bit 4 don't require CTS to transmit                ;
;                       bit 5 don't require DSR to transmit                ;
;                       bit 6 sets baud rate to 19200                      ;
;                       bit 7 sets baud rate to 38400                      ;
;                                                                          ;
;               ah = 5      remove excom, restore old vectors, release memory ;
;                                                                          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

int14:
        sti
        push    ds
        push    si
        push    dx
        push    cx
        push    bx
        ; point to selected com port control block with ds:si
        mov     bx, cs
        mov     ds, bx
        lea     si, pcb
        or      dx, dx
        jz      b00
        add     si, size pcbs
b00:
        ; get port address in dx, return if port not installed
        mov     dx, [si].pbase
        or      dx, dx
        jz      b40

        ; ah has request type, jump to selected routine
        or      ah, ah
        jz      b000                            ; initialize
        dec     ah
        jz      b100                            ; transmit a char
        dec     ah
        jnz     b10
        jmp     b200                            ; receive a char
b10:
        dec     ah
        jnz     b20
        jmp     b300                            ; get status
b20:
        dec     ah
        jnz     b30
        jmp     b400                            ; extended initialize
b30:
        dec     ah
        jnz     b40
        jmp     b500                            ; remove excom
b40:
        pop     bx
        pop     cx
        pop     dx
        pop     si
        pop     ds
        iret
        ;
        ; ah = 0
        ; initialize the port
b000:
        push    ax

        ; initialize the pcb, this empties the input buffers
        lea     ax, [si].buf
        mov     [si].putptr, ax
        inc     ax
        inc     ax
        mov     [si].getptr, ax
        add     ax, BUFSZ * 2
        mov     [si].bufend, ax
        sub     ax, ax
        mov     [si].bufcnt, ax
        mov     [si].rxoff, al

        ; assert flow control signals and enable port interrupts
        add     dx, 4
        mov     al, DTR or RTS or OUT2
        out     dx, al
        cli
        in      al, 021h
        and     al, [si].mask
        out     021h, al                        ; unmask com port on int controller
        sti
        sub     dx, 3
        mov     al, 1
        out     dx, al                          ; enable receive int on uart

        ; set up baud rates and character attributes
        add     dx, 2
        pop     ax
        mov     bl, al
        and     al, 01Fh
        mov     [si].lchar, al
        mov     al, [si].opts
        and     al, B19200 or B38400
        jz      b010
        call    b440                            ; init only character attributes
        jmp     short b020
b010:
        mov     cl, 4
        rol     bl, cl
        and     bx, 0Eh                         ; isolate baud rate selection
```

(continued on page 72)

COM PORT DRIVER

Listing One *(Listing continued, text begins on page 42.)*

```
            call      b430                          ; init baud and character attributes
b020:
            sub       dx, 3
            jmp       b300                          ; return status


            ;
            ; ah = 1
            ; transmit the character in al
b100:
            push      ax
            add       dx, 6
            mov       bh, [si].opts
            not       bh
            and       bh, NODSR or NOCTS
            ; optionally wait for DSR and/or CTS
            call      b130
            jnz       b110
            dec       dx
            mov       bh, 020h
            ; wait for the transmitter to be ready
            call      b130
            jnz       b110
            sub       dx, 5
            pop       ax
            push      ax
            ; actually send the character
            out       dx, al
            call      b310
            jmp       short b120
b110:
            ; restore al and indicate a timeout (bit 7 of status, ah)
            call      b310
            or        ah, 080h
b120:
            pop       cx
            mov       al, cl
            jmp       b40

            ;
            ; wait for selected status, bh is status, dx is port address
b130:
            mov       bl, [si].timeout              ; delay counter
b140:
            sub       cx, cx
b150:
            in        al, dx
            and       al, bh
            cmp       al, bh
            jz        b160                          ; status match
            loop      b150
            dec       bl
            jnz       b140
            or        bh, bh                        ; timeout, set non-zero
b160:
            ret


            ;
            ; ah = 2
            ; receive a character, put it in al
b200:
            cmp       [si].bufcnt, 0
            jne       b230                          ; chars in buffer, return one

            ; wait for a character to appear in the buffer, or return timeout
            mov       al, [si].timeout
            add       al, al                        ; shorter timeout loop, so loop more
b210:
            sub       cx, cx
b220:
            cmp       [si].bufcnt, 0
            jne       b230
            loop      b220
            dec       al
            jnz       b210
            mov       ax, 08000h                    ; timeout return value
            jmp       b40
b230:
            ; if receiver is disabled and room now exists, enable receiver
            cmp       [si].rxoff, 0
            je        b250                          ; receive is already enabled
            cmp       [si].bufcnt, XONSZ
            jg        b250                          ; buffer fuller that turn on point
            test      [si].opts, XNXF
            jz        b240
            mov       [si].rxoff, 0                 ; indicate receiver enabled
            mov       al, 'Q' - '@'
            out       dx, al                        ; send ^Q
b240:
            ; it's easier to just assert DTR & RTS rather than see if needed
            add       dx, 4
            in        al, dx
            or        al, DTR or RTS
            out       dx, al                        ; assert DTR and RTS
b250:
            ; get the char from the buffer and update buffer get pointer
            call      b310
            and       ah, 01Eh                      ; status reported with a receive
            inc       bx
            inc       bx
            cmp       bx, [si].bufend
            jne       b260
            lea       bx, [si].buf
b260:
```

```
        mov     [si].getptr, bx                 ; updated pointer
        dec     [si].bufcnt
        jmp     b40

        ;
        ; ah = 3
        ; get port status
b300:
        call    b310
        inc     dx
        in      al, dx                          ; modem status
        jmp     b40

        ;
        ; get line status from input buffer and/or hardware into ah
b310:
        mov     dx, [si].pbase
        add     dx, 5
        in      al, dx                          ; line status
        mov     ah, al
        cmp     [si].bufcnt, 0
        je      b320

        ; when chars in buffer, fix status as per status in buffer
        ; return as above plus bx = getptr, al = character
        mov     cl, al
        mov     bx, [si].getptr
        mov     ax, [bx]                        ; status:data (ah:al)
        mov     ch, 01Eh                        ; these status bits from buffer
        and     ah, ch
        not     ch
        and     cl, ch
        or      ah, cl
        or      ah, 1                           ; char in buffer, show data ready
b320:
        ret

        ;
        ; ah = 4
        ; extended initialization
b400:
        ; save the options in the pcb
        mov     [si].opts, al

        ; if an extended baud rate, set up the uart
        add     dx, 3
        test    al, B19200
        jz      b410
        mov     bx, baud19 - bauds
        call    b430
        jmp     short b420
b410:
        test    al, B38400
        jz      b420
        mov     bx, baud38 - bauds
        call    b430
b420:
        mov     ax, 05A5Ah                      ; magic value to identify excom
        jmp     b40

        ;
        ; init the baud rate and character attributes
b430:
        mov     al, 080h
        out     dx, al
        mov     ax, [bx+bauds]
        sub     dx, 3
        out     dx, al                          ; set low order divisor on uart
        inc     dx
        mov     al, ah
        out     dx, al                          ; set high order divisor on uart
        inc     dx
        inc     dx
b440:
        mov     al, [si].lchar
        out     dx, al                          ; set character attributes
        ret

        ;
        ; ah = 5
        ; remove excom
b500:
        push    ds
        push    ds

        ; restore status of interrupt controller and uarts
        cli
        mov     ah, oldmsk
        and     ah, C1MASK or C2MASK
        in      al, 021h
        and     al, not (C1MASK or C2MASK)
        or      al, ah
        out     021h, al
        sti
        lea     si, pcb                         ; com1 pcb
        call    b510                            ; restore com1 uart status
        lea     si, pcb + size pcbs  ; com2 pcb
        call    b510                            ; restore com1 uart status
        jmp     short b530

        ; subroutine to restore uart status
b510:
        mov     dx, [si].pbase
        or      dx, dx
        jz      b520                            ; no port
```

(continued on next page)

Listing One *(Listing continued, text begins on page 42.)*

```
            add         dx, 3
            mov         al, 080h
            out         dx, al                          ; set access to baud divisors
            sub         dx, 3
            mov         al, [si].olddll
            out         dx, al                          ; baud divisor low
            inc         dx
            mov         al, [si].olddlm
            out         dx, al                          ; baud divisor high
            add         dx, 2
            mov         al, [si].oldlcr
            out         dx, al                          ; line control register
            inc         dx
            mov         al, [si].oldmcr
            out         dx, al                          ; modem control register
            sub         dx, 3
            mov         al, [si].oldier
            out         dx, al                          ; interrupt enable register
b520:
            ret

b530:
            ; restore old int 0Bh vector
            lds         dx, old0B
            mov         ah, 025h
            mov         al, 0Bh
            int         021h

            ; restore old int 0Ch vector
            pop         ds
            lds         dx, old0C
            mov         ah, 025h
            mov         al, 0Ch
            int         021h

            ; restore old int 14h vector
            pop         ds
            lds         dx, old14
            mov         ah, 025h
            mov         al, 14h
            int         021h

            ; free excom's memory, this memory
            push        es
            mov         ax, cs
            mov         es, ax
            mov         ah, 049h
            int         021h
            pop         es
            jmp         b40

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                          ;
;           installation entry point, must be at file end                  ;
;                       initialize and stay resident                       ;
;                                                                          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
excom:

            ; initialize constant parts of each pcb
            mov         ax, 040h
            mov         es, ax                          ; bios data segment
            sub         di, di                          ; offset to com1 base address
            mov         bx, 07Ch                        ; offset to com1 timeout
            mov         cl, not C1MASK                  ; com1 interrupt mask
            lea         si, pcb                         ; com1 pcb address
            call        c00                             ; init com1
            inc         di
            inc         di                              ; offset to com2 base address
            inc         bx                              ; offset to com2 timeout
            mov         cl, not C2MASK                  ; com2 interrupt mask
            add         si, size pcbs                   ; com2 pcb address
            call        c00                             ; init com2
            jmp         short c10

            ; subroutine to initialize parts of a pcb
c00:
            mov         dx, es:[di]                     ; port base from bios
            mov         [si].pbase, dx
            mov         al, es:[bx]
            mov         [si].timeout, al                ; copy timout from bios
            mov         [si].mask, cl                   ; save interrupt mask
            ret

c10:
            ; save old status of interrupt controller and uarts
            in          al, 021h
            mov         oldmask, al
            lea         si, pcb                         ; com1 pcb
            call        c20                             ; save old com1 uart status
            lea         si, pcb + size pcbs  ; com2 pcb
            call        c20                             ; save old com1 uart status
            jmp         short c40

            ; subroutine to save uart status
c20:
            mov         dx, [si].pbase
            or          dx, dx
            jz          c30                             ; no port
            add         dx, 3
            in          al, dx
            and         al, 07Fh
            mov         [si].oldlcr, al                 ; line control register
            mov         al, 080h
            out         dx, al                          ; set access to baud divisors
            sub         dx, 3
```

```
        in      al, dx
        mov     [si].oldd11, al          ; baud divisor low
        inc     dx
        in      al, dx
        mov     [si].olddlm, al          ; baud divisor high
        add     dx, 2
        mov     al, [si].oldlcr
        out     dx, al                   ; restore register access
        inc     dx
        in      al, dx
        mov     [si].oldmcr, al          ; modem control register
        sub     dx, 3
        in      al, dx
        mov     [si].oldier, al          ; interrupt enable register
c30:
        ret

c40:
        ; release environment memory, not used
        mov     bx, 02Ch
        mov     ax, [bx]
        mov     es, ax                   ; address of env
        mov     ah, 049h
        int     021h                     ; free memory

        ; save old int 0Bh vector, install new handler
        push    ds
        mov     ax, cs
        mov     ds, ax
        mov     ah, 035h
        mov     al, 0Bh
        int     021h
        mov     word ptr old0B, bx
        mov     ax, es
        mov     word ptr old0B + 2, ax
        lea     dx, int0B
        mov     ah, 025h
        mov     al, 0Bh
        int     021h

        ; save old int 0Ch vector, install new handler
        mov     ah, 035h
        mov     al, 0Ch
        int     021h
        mov     word ptr old0C, bx
        mov     ax, es
        mov     word ptr old0C + 2, ax
        lea     dx, int0C
        mov     ah, 025h
        mov     al, 0Ch
        int     021h

        ; save old int 14h vector, install new handler
        mov     ah, 035h
        mov     al, 014h
        int     021h
        mov     word ptr old14, bx
        mov     ax, es
        mov     word ptr old14 + 2, ax
        lea     dx, int14
        mov     ah, 025h
        mov     al, 014h
        int     021h
        pop     ds

        ; exit and keep everything above the entry point 'excom'
        lea     dx, excom
        mov     cl, 4
        shr     dx, cl
        inc     dx
        mov     ah, 031h
        mov     al, 0
        int     021h                     ; terminate-and-stay-resident

_text   ends
        end     start
        endm
```

End Listing One

Listing Two

```
#include  <stdio.h>
#include  <dos.h>

/*
          (C) 1987, Crystal Computer Consulting Inc.
          This software may be used freely, at your own risk,
          as long as this notice is not removed.
*/

#define   EXINIT     4              /* ah value for extended init */
#define   EXCOM      0x5A5A         /* magic value to identify excom */
#define   DTR        0x01           /* DTR input flow control */
#define   RTS        0x02           /* RTS input flow control */
#define   XNXFIN     0x04           /* XON/XOFF input flow control */
#define   NOCTS      0x10           /* CTS not required for transmit */
#define   NODSR      0x20           /* DSR not required for transmit */
#define   B19200     0x40           /* set baud rate to 19200 */
#define   B38400     0x80           /* set baud rate to 38400 */
#define   COMINIT    0x100          /* port was selected */

typedef struct {
          char
                     *str;          /* command string */
          void
                     (*fnct)();     /* command to execute */
          int
                     arg;           /* argument to pass */
} CMDS;                             /* exmode command table */

char
          *insmsg = "excom installed",
          *remmsg = "excom not installed",
          *helpmsg;
```

(continued on next page)

Listing Two *(Listing continued, text begins on page 42.)*

```
int
        init1 = 0,                      /* extended init for com1 */
        init2 = 0,                      /* extended init for com2 */
        portnum = -1;                   /* port to initialize */
void
        exit(), install(), remove(), exinit(), setport();
CMDS
        cmds[] = {
                "install", install,  0,
                "remove",  remove,   0,
                "com1",    setport,  0,
                "com2",    setport,  1,
                "dtr",     exinit,   DTR,
                "rts",     exinit,   RTS,
                "xnxfin",  exinit,   XNXFIN,
                "nocts",   exinit,   NOCTS,
                "nodsr",   exinit,   NODSR,
                "19200",   exinit,   B19200,
                "38400",   exinit,   B38400,
                NULL
        };


main(argc, argv)
int
        argc;
register char
        **argv;
{
        register CMDS
                *cmdp;

        /* is excom installed */
        if (int14(EXINIT, 0, 0) == EXCOM && int14(EXINIT, 0, 1) == EXCOM)
                helpmsg = insmsg;
        else
                helpmsg = remmsg;

        if (argc == 1)
                help(helpmsg);

        while (*++argv != NULL) {
                /* look up the argument in the command table */
                for (cmdp = cmds;  cmdp->str != NULL;  ++cmdp)
                        if (strcmp(*argv, cmdp->str) == 0)
                                break;

                if (cmdp->str == NULL)
                        help("bad command");

                else {
                        /* excom must be installed to set options */
                        if (helpmsg == remmsg && cmdp->fnct != install)
                                help(helpmsg);

                        /* execute the command */
                        (*cmdp->fnct)(cmdp->arg);
                }
        }

        if ((init1 & (B19200 | B38400)) == (B19200 | B38400)
            || (init2 & (B19200 | B38400)) == (B19200 | B38400))
                help("ambiguous baud rate setting");

        /* actually send the init bits to excom */
        if ((init1 & COMINIT) != 0)
                int14(EXINIT, init1, 0);

        if ((init2 & COMINIT) != 0)
                int14(EXINIT, init2, 1);

        exit(0);
}

/* install excom */
void
install()
{
        system("excom");
        helpmsg = insmsg;
}


/* remove excom */
void
remove()
{
        int14(5, 0, 0);
        helpmsg = remmsg;
}


/* collect up the init bits for each port */
void
exinit(thebit)
int
        thebit;
{
        if (portnum == -1)
                help("no port selected");

        else if (portnum == 0)
                init1 |= thebit;

        else
```

```
setport(newport)
int
          newport;
{
          portnum = newport;

          if (portnum == 0)
                    init1 |= COMINIT;

          else
                    init2 |= COMINIT;
}


/* perform int 14h with ah, al & dx as passed, return ax value */
int14(cmd, val, port)
int
          cmd,
          val,
          port;
{
          union REGS
                    ir,                 /* registers send to bios */
                    or;                 /* registers returned from bios */

          ir.h.ah = cmd;
          ir.h.al = val;
          ir.x.dx = port;

          int86(0x14, &ir, &or);

          return or.x.ax;
}


/* provide a bit of assistance */
help(msg)
char
          *msg;
{
          printf("\n%s\n\n", msg);

          fputs("install\t\tinstall excom\n", stdout);
          fputs("remove\t\tremove excom\n", stdout);
          fputs("com1\t\tsubsequent commands for com1\n", stdout);
          fputs("com2\t\tsubsequent commands for com2\n", stdout);
          fputs("dtr\t\tuse DTR for input flow control\n", stdout);
          fputs("rts\t\tuse RTS for input flow control\n", stdout);
          fputs("xnxfin\t\tuse XON/XOFF (^S ^Q) input flow control\n", stdout);
          fputs("nocts\t\tdon't require CTS to transmit\n", stdout);
          fputs("nodsr\t\tdon't require DSR to transmit\n", stdout);
          fputs("19200\t\tset baud rate\n", stdout);
          fputs("38400\t\tset baud rate\n", stdout);

          exit(1);
}
```

End Listings

Listing One *(Text begins on page 50.)*

```
Listing 1: MemOvrly.Inc

(***************************************************************
 *                                                             *
 *        Turbo Pascal Memory Overlay Routines                 *
 *                                                             *
 *        Copyright (C) 1986 by Steve McMahon                  *
 *                                                             *
 *        All Rights Reserved.                                 *
 *                                                             *
 ***************************************************************)

 (*

 Limitations:

 These routines have been tested only for Turbo 3.01A (both
 PC-DOS and generic MS-DOS).  They may not work under 3.0
 (the celebrated FileSize bug may cause trouble) and will
 certainly not work under 2.0XX.

 Memory overlay files must be < 64k in size!

 NORMAL overlays nested inside memory overlays should work, but
 trying to nest memory overlays inside memory overlays would
 be disasterous!

 OvrPath will not work in conjunction with memory overlays!
 (Writing a replacement routine would be simple if the code
 below makes sense to you.)

 I/O testing in InitOverlay is just Turbo's Native.  Anyone
 really needing memory overlays will probably wish to install
 their own I/O error checking.

 *)

CONST
  RequiredHeap = $1000;   {Paragraphs of Heap Required by Program
                           for other purposes than memory overlays.
                           Change this to suit your needs for
                           dynamic storage.}

TYPE
  {Type used in both InitOverlay and DisposeOverlayStorage}
  OverlayProcedure = RECORD
                  CASE Boolean OF
                    True :
                      ( OldCall    : ARRAY[1..3] OF Byte;
                        OldOffset  : Integer;
                        FileName   : ARRAY[1..13] OF Char;
                      );
                    False :
                      ( NewCallInstruction : ARRAY[1..3]OF Byte;
                        NewCallAddress     : Integer;
                        CurrentOffset      : Integer;
                        OverlayCodeLoc     : ^Byte;
                        NewRoutineLoc      : Integer;
                        OverlaySize        : Integer;
                      )
                  END;

PROCEDURE NewOverlayHandler;
  BEGIN
    INLINE(
      {When this routine receives control, AX contains the
      number of bytes in the desired overlay & BX contains the
      offset (in pages) of the desired overlay within the
      overlay file (now on the heap).}

      {First, check to see if the desired overlay is already in
      place by comparing DX with the offset recorded in memory
      immediately after the call instruction.  If they match,
      no load is necessary}

      $5E/                  {POP     SI          }
      $2E/$3B/$14/          {CMP     DX,CS:[SI]  }
      $74/$1B/              {JZ      RUN_OVERLAY}

      {Save vital registers}
      $56/                  {PUSH    SI          }
      $1E/                  {PUSH    DS          }

      {Load ES:DI with destination address (the point the
      code will run at).  Displace to account for header.}
      $0E/                  {PUSH    CS          }
      $07/                  {POP     ES          }
      $8B/$FE/              {MOV     DI,SI       }
      $83/$C7/$0D/          {ADD     DI,0DH      }

      {Fetch heap address of source overlay code from memory
      position two bytes after first byte after call to this
      routine.  Store it in DS:SI}
      $46/                  {INC     SI          }
      $46/                  {INC     SI          }
      $2E/$C5/$34/          {LDS     SI,CS:[SI]  }

      {Multiply overlay page by 100H to get number of bytes code
      is displaced from start of overlay code area (on heap).
      Add to source offset in SI.}
      $8A/$F2/              {MOV     DH,DL       }
      $32/$D2/              {XOR     DL,DL       }
      $03/$F2/              {ADD     SI,DX       }

      {Put number of bytes to move in CX}
      $8B/$C8/              {MOV     CX,AX       }

      {Copy CX bytes from DS:SI to ES:DI}
```

```
        $FC/                      {CLD                  }
        $F3/$A4/                  {REPZ     MOVSB       }

        {Recover mauled registers}
        $1F/                      {POP      DS          }
        $5E/                      {POP      SI          }

        {RUN_OVERLAY:}
        $83/$C6/$0D/              {ADD      SI,0DH      }
        $FF/$E6                   {JMP      SI          }
        );
  END;


PROCEDURE InitOverlay(OverlayCallOffset : Integer);
  VAR
    OverlayCallPtr : ^OverlayProcedure;
    TestSize, i    : Integer;
    s              : STRING[13];
    f              : FILE;
  BEGIN
    OverlayCallPtr := Ptr(CSeg, OverlayCallOffset);
    WITH OverlayCallPtr^ DO
      BEGIN
        {Obtain overlay file name}
        i := 1;
        s := '';
        WHILE FileName[i] <> #0 DO
          BEGIN
            s := s + FileName[i];
            i := i + 1;
          END;
        {Open overlay file as untyped file}
        Assign(f, s);
        Reset(f);
        {determine file size in $80-byte sectors}
        TestSize := FileSize(f);
        {Check to see if there's enough space on the heap.}
        {If there isn't, leave the overlay on disk}
        IF (MemAvail > (RequiredHeap + TestSize * 8)) AND
           (MaxAvail >= TestSize * 8) THEN {there's enough space}
          BEGIN                 {install overlay}
            OverlaySize := TestSize;
            GetMem(OverlayCodeLoc, OverlaySize * $80);
            BlockRead(f, OverlayCodeLoc^, OverlaySize, i);
            NewCallInstruction[1] := $2E; {CS:}
            NewCallInstruction[2] := $FF;
            NewCallInstruction[3] := $16; {indirect near call}
            NewCallAddress := Ofs(NewRoutineLoc);
            NewRoutineLoc := Ofs(NewOverlayHandler) + 7;
            {extra 7 bytes skips turbo's procedure overhead}
            CurrentOffset := $FFFF; {force load on first call}
          END;
        Close(f);
      END;
  END;


PROCEDURE DisposeOverlayStorage(OverlayCallOffset : Integer);
  VAR
    OverlayCallPtr : ^OverlayProcedure;
  BEGIN
    OverlayCallPtr := Ptr(CSeg, OverlayCallOffset);
    WITH OverlayCallPtr^ DO
      IF NewCallInstruction[3] = $16 THEN {Overlay is in memory}
        FreeMem(OverlayCodeLoc, OverlaySize * $80);
  END;
```

End Listings

Listing One *(Text begins on page 54.)*

```
list


TPATH=/u/bbs/rbin

# check the command argument - $1

if [ "$1" = MS]
then

# argument is for MS-DOS files

    echo 'who am i | cut -f1 -d" " 'date | cut -c1-16 ' "MS.list" >>/u/bbs/log.file
    echo
    more $TPATH/MS.list

elif [ "$1" = Mac ]
then

# argument is for Macintosh files

    echo 'who am i | cut -f1 -d" " 'date | cut -c1-16 ' "Mac.list" >>/u/bbs/log.file
    echo
    more $TPATH/Mac.list

elif [ "$1" = Unix ]
then

# argument is for Unix files

    echo 'who am i | cut -f1 -d" " 'date | cut -c1-c16 ' "Unix.list" >>/u/bbs/log.
                                                                              file
    echo
    more $TPATH/Unix.list
else

# no argument was entered

    echo
    echo "List which directory? (MS, Mac, or Unix)"
fi
```

End Listing One

Listing Two

```
dwnld


# identify which file directory (contained in the first argument - $1)

if [ "$1" = MS ]
then

# an MS-DOS file

    filedir=/u/bbs/MS-files
elif [ "$1" = Mac ]
then

# a Macintosh file

    filedir=/u/bbs/Mac-files
elif [ "$1" = Unix ]
then

# a Unix file

    filedir=/u/bbs/Unix-files
else

# no valid file directory was entered

    echo "Follow the dwnld command with a file directory - MS, Mac, or Unix"
fi

# verify that a file name has been entered (contained in the second argument - $2)

if [ -n "$2" ]
then

# verify that the file name exists within the selected directory

    if [ -f "$filedir/$2" ]
    then
        echo 'who am i | cut -f1 -d" " 'date | cut -c1-16 ' "dwnld" $1 $2
                                                       >>/u/bbs/log.file

# select a file transfer protocol

        echo "Transfer potocol (X = Xmodem; A = ASCII): \c"
        read method
        if [ "$method" = X -o "$method" = x ]
        then

# send a binary file using the XModem protocol

            xmodem -sb $filedir/$2
        else

# send an ASCII file
            echo "Prepare to transfer file.  Press return to start."
            read dummy
            cat $filedir/$2
            sleep 5
            echo "Download complete.  Press return to continue."
```

```
              read dummy
          fi
      else

    # a valid file name wasn't entered

          echo "Sorry, that file name does not exist.  Be sure to type the"
          echo "file name exactly as it appears in the directory listing."
          echo "Upper and lower case letters are different."
      fi
 else

    # no file name was entered

      echo "Please enter a file name after the directory on the command line."
      echo "Use the list command to see available file names."
 fi
```

End Listing Two

Listing Three

```
upld


    # verify that a file name has been entered (in argument 1 - $1)

 if [ -n "$1" ]
 then

    # verify that a file with the same name isn't already stored in the Uploads directory

      if [ -f /u/bbs/Uploads/$1 ]
      then

    # the file name already exists
          echo "Please give your file another name.  $1 already exists."
      else

    # collect the file transfer protocol

          echo 'who am i | cut -f1 -d" " ' 'date | cut -c1-16' "upld" $1 >>/u/bbs/log.
                                                                                file
          echo "Transfer protocol (X = Xmodem; A = ASCII): \c"
          read method
          if [ "$method" = X -o "$method" = x ]
          then

    # receive a binary file using the XModem protocol

              xmodem -rb /u/bbs/Uploads/$1
          else

    # receive an ASCII file

              echo
              echo "Begin sending ASCII file."
              echo "Type a CNTRL-d when finished transmitting."
              echo
              cat > /u/bbs/Uploads/temp
              cp /u/bbs/Uploads/temp /u/bbs/Uploads/$1
          fi

    # collect the file description

          echo "Please enter a one line description of your file."
          read desc
          echo $1 $desc >>/u/bbs/Uploads/Doc.file
      fi
 else
    # no file name was entered

      echo "Please enter the name of the file you wish to upload on"
      echo "the command line."
 fi
```

End Listing Three

Listing Four

```
readmsg


 FIRST=/u/bbs/.first
 LAST=/u/bbs/.last
 MSG=/u/bbs/msg
 echo 'who am i | cut -f1 -d" " ' ' date | cut -c1-16' "readmsg" >>/u/bbs/log.file
 echo

    # display the number of the first and last messages available

 echo "Messages available:"
 cat $FIRST
 echo "through"
 cat $LAST
 echo

    # accept the number of the first message to be displayed

 echo "Message number to read ('q' to quit): \c"
 read message

    # loop until the user wants to quit

 while [ "$message" != q ]
 do
      echo

    # verify that the message number entered actually exists
```

(continued on next page)

Listing Four *(Listing continued, text begins on page 54.)*

```
    if [ -f $MSG/$message ]
    then

# display the message

        cat $MSG/$message
    elif [ "$message" != q ]
    then

# the entered message number doesn't exist

        echo "Sorry.  That message doesn't exist."
    fi
    echo

# accept the next message number to read

    echo "Message number to read: ('q' to quit): \c"
    read message
done
echo
```

End Listing Four

Listing Five

```
send


echo 'who am i | cut -f1 -d" " ' 'date | cut -c1-16' "send" >>/u/bbs/log.file
SCAN=/u/bbs/msg/.index
LAST=/u/bbs/.last
MSG=/u/bbs/msg
L='cat $LAST'
D='date'

# increment the number of the last message entered

L='expr $L + 1'

# save the new "last message" value

echo $L >$LAST
echo

# collect message header information

echo "To: \c"
read to
echo "Subject: \c"
read subject
echo "From: \c"
read who
trap 'rm -f $MSG/$L;continue' 2 3
ech
echo "Start typing your message.  You have 20 lines available."
echo "Type a period on a new line to end the message."
echo

# store the message header in the message file

echo "# $L  From: $who  $D" >$MSG/$L
echo "  To: $to" >>$MSG/$L
echo "  Subject: $subject" >>$MSG/$L
echo >>$MSG/$L

# collect the body of the message

NL=0
read newline
while [ "$newline" != "." -a "$NL" -lt 20 ]
do
    echo $newline >>$MSG/$L
    NL='expr $NL + 1'
    read newline
done
echo

# update the message index for use by the scan command

echo "Message being posted, please wait... \c"
echo "# $L From: $who  $D" >>$MSG/temp
echo "  To: $to" >>$MSG/temp
echo "  Subject: $subject" >>$MSG/temp
echo " " >>$MSG/temp
cat $SCAN >>$MSG/temp
mv $MSG/temp $SCAN
echo
```

End Listing Five

Listing Six

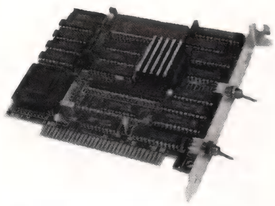
```
pmail


USERS=/u/bbs/rbin/user.file
echo 'who am i | cut -f1 -d" " ' 'date | cut -c1-16' "pmail" >>/u/bbs/log.file

# display pmail instructions

echo
echo"  ************** Private Mail ****************"
echo
echo        Commands:
echo    s - send mail to another user
```

(continued on page 84)

MICROWAY ACCELERATES YOUR PC!

FastCACHE-286™

Runs your PC Faster than an AT!
Runs the 80286 at 9 or 12 MHz and the 80287 at 8, 9 or 12 MHz. Includes 8 kbytes of 55ns CACHE.



Compatible with Leading Edge Model D, Compaq, and Turbo motherboards. Includes 8088 Reboot Switch, DCache, Print Spooler and Diagnostics . . . **From $399**

LOTUS/INTEL EMS SPECIFICATION BOARDS

MegaPage™ The only EMS board which comes populated with two megabytes of cool-running, low power drain CMOS RAM installed. Includes RAM disk, print spooler, disk cache and EMS drivers. For the IBM PC, XT and compatibles...**$549**

MegaPage with ∅K. **$149**

MegaPage with 2 megabytes of HMOS RAM. **$419**

MegaPage AT/ECC™ EMS card for the PC AT and compatibles includes Error Correction Circuitry. With ECC, 11 RAM chips cover 256K so the user never encounters RAM errors. With 1 megabyte CMOS RAM. **$699**

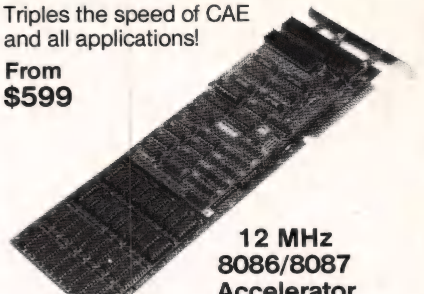
INTEL, JRAM, or Maynard . . . . . CALL

INTEL INBOARD 386 ∅K. $1325

NUMBER SMASHER/ECM™

Triples the speed of CAE and all applications!

From $599



12 MHz 8086/8087 Accelerator Plus A Megabyte for DOS!

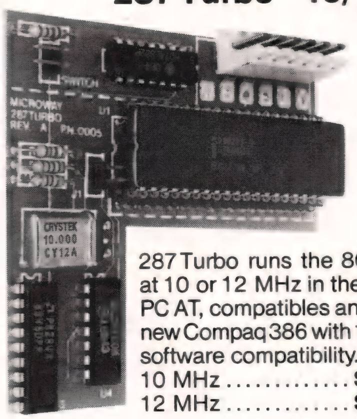
For the IBM PC, XT and compatibles

***PC Magazine* "Editor's Choice"**

8087 SOFTWARE

IBM BASIC COMPILER	$465
MICROSOFT QUICK BASIC	$79
87BASIC COMPILER PATCH	$150
87BASIC/INLINE	$200
IBM MACRO ASSEMBLER	$155
MS MACRO ASSEMBLER	$99
87MACRO/DEBUG	$199
MICROSOFT FORTRAN V4	$299
RM FORTRAN	$399
LAHEY FORTRAN F77L	$477
MS or LATTICE C	CALL
STSC APL★PLUS/PC	$450
STSC STATGRAPHICS	$675
SPSS/PC+	$695
87SFL Scientific Functions	$250
87FFT	$200
OBJ → ASM	$200
PHOENIX PRODUCTS	CALL

287 Turbo™-10/12



287 Turbo runs the 80287 at 10 or 12 MHz in the IBM PC AT, compatibles and the new Compaq 386 with 100% software compatibility.

10 MHz **$450**
12 MHz **$550**

***PC Magazine* "Editor's Choice"**

8087 UPGRADES

All MicroWay 8087s include a one year warranty, complete MicroWay Test Program and installation instructions.

8087 5 MHz $105
For the IBM PC, XT and compatibles

8087-2 8 MHz $154
For Wang, AT&T, DeskPro, NEC, Leading Edge

80287-3 5 MHz $179
For the IBM PC AT and 286 compatibles

80287-6 6 MHz $229
For 8 MHz AT and compatibles

80287-8 8 MHz $259
For the 8 MHz 80286 accelerator cards and Compaq 386

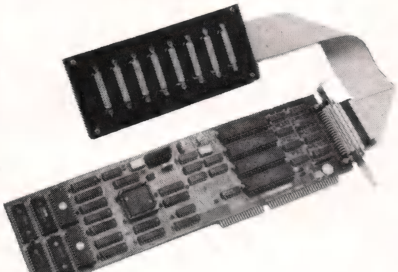
80287-10 10 MHz $395

PC-PAL™ Programmer $395

Call for great prices on V20, V30, 64K, 128K and 256K RAM

AT8™

Turns your AT into a high speed, multi-user Xenix business system!



8 port, intelligent serial controller with 3% response degradation. Includes 8 MHz 80186 with built in DMA **$1299**

MICROWAY SOFTWARE FOR LOTUS 1-2-3™

FASTBREAK™ employs the 8087 to increase the speed of Lotus 1-2-3™ Version 1A or 1A*. Users are reporting speed ups of between 3 and 36 to 1. When run with our NUMBER SMASHER accelerator card, recalculation speed ups of 10 to 30 are being reported **$79**

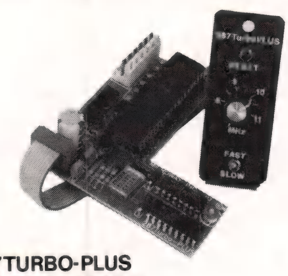
PowerDialer® Add-In for Lotus 1-2-3 Release 2. Automated telephone dialing from within 1-2-3. Adds least cost routing, automatic carrier selection and automated phone book worksheet. Builds customized dialing applications. Can be used with DesqView **$79**

HOTLINK™ adds easy linking of spreadsheets to Lotus 1-2-3 Version 1A. . **$99**

287 TURBO-PLUS™

Speeds up your AT

Adjustable 80286 Clock 6-12 MHz
10 MHz 80287 Clock
Plus Full Hardware Reset. **$149**
Optional 80286-10 . . . . . **$175**



287TURBO-PLUS
With 80287 10 MHz. **$549**
With 80287 12 MHz. **$629**

CALL (617) 746-7341 FOR OUR COMPLETE CATALOG

MicroWay
P.O. Box 79
Kingston, Mass.
02364 USA
(617) 746-7341

You Can Talk To Us!

MicroWay Europe
32 High Street
Kingston-Upon-Thames
Surrey England KT1 1HL
Telephone: 01-541-5466

CIRCLE 300 ON READER SERVICE CARD

Listing Six *(Listing continued, text begins on page 54.)*

```
echo    r - read mail sent to you
echo    l - list the users by user id and name
echo    x - exit the private mail system
echo

# accept the first pmail command

echo "Command: \c"
read choice

# loop until the user wants to quit

while [ "$choice" != x -o "$choice" != X ]
do
    echo
    case $choice in
        [Ss])

# send private mail
# print instructions

            echo "WARNING: To successfully send mail observe these rules:"
            echo
            echo "   1. If you wish to erase a character, use the backspace key."
            echo "DO NOT, DO NOT use the delete key."
            echo "   2. Type a carriage return at the end of each line on the"
            echo "screen.  Lines should be no more than 80 characters."
            echo "   3. Do not use the following characters in your message (they"
            echo "have special meaning to the system): @ and #"
            echo
            echo "Press RETURN to begin sending your message: \c"
            read dummy
            echo

# collect user id of recipient

            echo "To whom (user id): \c"
            read to

# verify that recipient entered is a valid user id

            TPERS='grep $to $USERS | wc -c'
            if [ "$TPERS" -eq 0 ]
            then

# user id is invalid

                echo
                echo "Sorry, that user id doesn't exist."
                echo "List them with the l command."
            else

# send mail

                echo
                echo "Start typing your message.  Type a period on a new"
                echo "line to send the message."
                echo
                mail $to
            fi

# collect next option

            echo
            echo "Command (x to exit): \c"
            read choice
            continue;;
        [Rr])

# read private mail
# print instructions

            echo "After each message you will receive a '?' prompt."
            echo "Type 'd' to delete the message, <cr> to leave it in"
            echo "your mailbox, or 'q' to stop reading mail."
            echo

# real the mail
            mail
            echo

# collect the next command

            echo "Command (x to exit): \c"
            read choice
            continue;;
        [Ll])

# display a list of user names and associated user id's

            echo
            more $USERS
            echo

# collect the next command

            echo "Command (x to exit): \c"
            read choice
            continue;;
        *)

# if an unrecognized command was entered, prompt for another one

            echo "Command (x to exit): \c"
            read choice
            continue;;
    esac
done
echo
```

End Listing Six

(Listing Seven begins on page 86.)

EVEN MORE POWER AND FLEXIBILITY

BRIEF 2.0

Users and industry press alike have unanimously proclaimed BRIEF as the best program editor available today. Now, the best gets better, with the release of BRIEF 2.0.

Straight from the box, BRIEF offers an exceptional range of features. Many users find that BRIEF is the only editor they'll ever need, with features like real, multi-level Undo, flexible windowing and unlimited file size. But BRIEF has tremendous hidden power in its exclusive macro language. With it, you can turn BRIEF into your own custom editor containing the commands and features you desire. It's fast and easy.

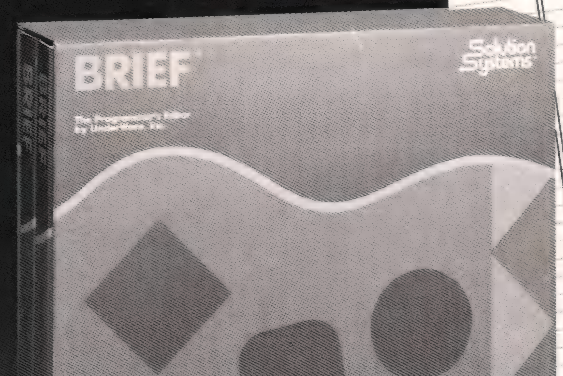
Jerry Pournelle, columnist for BYTE magazine summed it all up by saying BRIEF is, "**Recommended. If you need a general purpose PC programming editor, look no further.**" His point of view has been affirmed by rave reviews in C JOURNAL, COMPUTER LANGUAGE, DR. DOBB'S JOURNAL, DATA BASED ADVISOR, INFOWORLD AND PC MAGAZINE.

One user stated "BRIEF is one of the few pieces of software that I would dare call a masterpiece." Order BRIEF now and find out why. **BRIEF 2.0 is just $195. If you already own BRIEF, call for upgrade information.**

TO ORDER CALL: 1-800-821-2492 (in MA call 617-659-1571)

As always, BRIEF comes with a 30 day money-back satisfaction guarantee.

Solution Systems™
335 Washington St.
Norwell, MA 02061
(617) 659-1571



Look at these BRIEF 2.0 enhancements!

Main Features:

- All new documentation with tutorials on basic editing, regular expressions **and** the BRIEF Macro Language.
- Setup program for easy installation and configuration. (Requires no knowledge of the macro language)
- Increased speed for sophisticated operations like Undo and Regular Expression Search.
- Expanded regular expressions, with matching over line boundaries.
- More block types, with marking by character, line or column.
- Command line editing (move cursor, add and delete characters, specify command parameters).
- Support for more programming languages.
- Optional borderless windows.
- Enhanced large display support, including wider displays.
- Reconfigurable indenting for C files (supports most indenting styles).

Plus the basic features that made BRIEF **SO** popular!

Basic Features:

- Full multi-level Undo
- Windows
- Edit many files at once
- File size limited only by disk space
- Automatic language sensitive indentation

Requires an IBM PC or compatible with at least 192K RAM.

CIRCLE 142 ON READER SERVICE CARD

Listing Seven *(Listing continued, text begins on page 54.)*

```
.profile from the Account new


     # set default path to an empty rbin directory

     PATH=/u/bbs/new/rbin

     # set level one prompt to remind users how to log off

     PS1='Type CNTRL-D now... '

     # display the welcome message

     cat msg1
     echo

     # make sure the user really wants an account

     echo "Do you wish to request a login? (Y/N) \c"
     read choice

     # continue only if the user answers yes

     if [ "$choice" = y -o "$choice" = Y ]
     then
         echo
         OK="N"

     # loop until user is satisfied with data

         while [ "$OK" != Y -a "$OK" != y ]
         do

     # collect user name, address, etc.

             echo "Enter your real name: \c"
             read realname
             echo
             echo "Enter the first line of your mailing address: (4 lines are available)"
             read address1
             echo
             echo "Enter the second line of your mailing address: <cr> if none"
             read address2
             echo
             echo "Enter the third line of your mailing address: <cr> if none"
             read address3
             echo
             echo "Enter the fourth line of your mailing address: <cr> if none"
             read address4
             echo
             echo "Enter your voice telephone number: \c"
             read phone
             echo

     # display the data entered for the user

             echo "This is what you have entered:"
             echo
             echo $realname
             echo
             echo $address1
             echo $address2
             echo $address3
             echo $address4
             echo
             echo $phone
             echo

     # ask the user to verify the data

             echo "Is this correct? (Y/N) \c"
             read OK
         done
         OK="N"

     # loop until user is happy

         while [ "$OK" != Y -a "$OK" != y ]
         do

     # collect the account data

             echo
             echo "Enter a one word login name: \c"
             read lognmae
             echo
             echo "Enter an initial password: \c"
             read initpasswd
             echo

     # display the entered data for the user

             echo "You have entered:"
             echo
             echo $logname
             echo $initpasswd
             echo

     # ask the user to verify the data

             echo "Is this correct?  (Y/N) \c"
             read OK
         done
         echo

     # write the new account data to a holding file
```

(continued on page 88)

Listing Seven *(Listing continued, text begins on page 54.)*

```
     echo $realname >>signups
     echo $address1 >>signups
     echo $address2 >>signups
     echo $address3 >>signups
     echo $address4 >>signups
     echo $phone >>signups
     echo $logname >>signups
     echo $initpasswd >>signups
     echo >>signups

  # tell the user that he or she is finished

     echo "This completes the login request process.  You'll receive an account"
     echo "confirmation through the mail within a week."
     echo
fi

  # tell the user to log off

echo
echo "Enter CTNRL-D to log off the system."
echo
```

End Listing Seven

Listing Eight

```
.profile for the info Account


  # set the default path to an empty rbin directory

PATH=/u/bbs/info/rbin

  # set the level one prompt to the logoff message

PS1='Type CNTRL-D now...'

  # display the welcome message

echo
echo "             Welcome to Scholastech Telecommunications"
echo "                      Scholastech Info"
echo
cat msg
echo
echo "Press the carriage return to begin: \c"
read dummy

  # display the contents of the information file

more info.file
echo
echo "Press the carriage return when done: \c"
read dummy
echo

  # give the user the choice of whether or not to sign up for workshops

echo "Do you wish to sign up for a workshop? (Y/N) \c"
read dummy

  # loop until user doesn't want to register for any more workshops

while [ "$dummy" = y -o "$dummy" = Y ]
do
     echo
     echo "Enter your name: \c"
     read regname
     echo
     echo "For which workshop are you registering? \c"
     read workshop
     echo
     echo
     echo "Enter your school or company name: \c"
     read school
     echo
     echo "For how many people are you registering? \c"
     read people
     echo
     echo "Enter a voice phone number where you can be reached in case"
     echo "there are any questions about your registration: \c"
     read phone
     echo

  # write the registration data to disk

     echo $regname >>signups
     echo $workshop >>signups
     echo $school >>signups
     echo $people >>signups
     echo $phone >>signups
     echo >>signups

  # allow the user to register for more than one workshop

     echo "Do you wish to register for another workshop? (Y/N) \c"
     read dummy
done

  # tell the user how to log off

echo
echo "Type CNTRL-D to log off the system."
echo
```

End Listing Eight

(Listing Nine begins on page 90.)

Listing Nine *(Listing continued, text begins on page 54.)*

```
logs


    # create a temporary file with a log on record for each remote user

who /etc/wtmp | grep ph >/u/user.log

    # if an "a" was entered as an argument to the command, display the entire log files

if [ "$1" = a ]
then
     echo
     echo
     echo "System Logins"
     echo

    # display all system use since /etc/wtmp was last purged

     more /u/user.log
     echo
     echo "Next?"
     read dummy
     echo

    # display all BBS use since /u/bbs/log.file was last purged

     echo "BBS User Log"
     echo
     more /u/bbs/log.file
     echo

    # otherwise, show only the last ten entries in each log file

else
     echo
     echo
     echo "Last System Logins"
     echo
     tail /u/user.log
     echo
     echo
     echo "Last BBS Activity"
     echo
     tail /u/bbs/log.file
     echo
fi

    # remove the temporary file

rm /u/user.log
```

End Listing Nine

Listing Ten

```
usage


    # print headings to report file

echo "User       Date      Login      Logout" >usage.temp
echo "----------------------------------------------" >>usage.temp
echo " " >>usage.temp

    # get a log in record from /etc/wtmp for remote users on device ph1

who /etc/wtmp | grep ph1 >temp1

    # get the user name

cut -f1 -d" " temp1 >temp2

    # get the log in date and time

cut -c25-36 temp1 >temp3

    # get log out record from /etc/wtmp and extract the log out time

who -d /etc/wtmp | grep ph1 | grep -v LOGIN | grep -v rc | cut -c32-36 >temp4

    # paste together for the report

paste temp2 temp3 temp4 >>usage.temp

    # print the report

pr usage.temp >usage.summary

    # remove temporary files

rm temp1
rm temp2
rm temp3
rm temp4
rm usage.temp

    # format the BBS command use log for output and purge the log file

pr /u/bbs/log.file >log.summary
> /u/bbs/log.file

    # clean out /etc/wtmp

> /etc/wtmp

    # send both reports to the printer

lp usage.summary
lp log.summary
```

End Listings

These softstrips by Cauzin Systems contain the listings for C Chest. The listings begin on page 94.

1 2 3 4 5 6

7 8

Listing One *(Text begins on page 102.)*

```c
#include <stdio.h>

/* PQ.C         General-purpose priority-queue routines.
 *              (C) 1987 Allen I. Holub. All Rights Reserved.
 *
 * typedef char *PQ;     Dummy typedef for priority queue.
 *
 * PQ   *pq_create( numele, elesize, cmp, swap, initheap )
 * int  numele;          Max # of elements in the queue
 * int  elesize;         Size of one element in bytes
 * int  (*cmp)();        Pointer to comparison function
 * int  (*swap)();       Pointer to swap function
 * char *initheap;       Inital heap or NULL to allocate
 *
 * pq_ins( p, item )     Insert Item into queue
 * PQ   *p;              Pointer to priority queue
 * char *item;           Pointer to item to insert
 *                       Return number of empty slots
 *                       before insertion (0 if none).
 *
 * int  pq_del( p, target ) Delete item from queue
 * PQ   *p;                 Pointer to priority queue
 * char *target;            Pointer to place to put deleted
 *                          item.
 *                          Return # of items in queue before
 *                          delete (0 if nothing deleted).
 *
 * char *pq_look( queue )   Look at (don't delete) top element
 * PQ   *queue;             Pointer to queue.
 *
 * int  pq_numele( queue )  Return # of elements in queue.
 * PQ   *queue;             Pointer to queue.
 *
 *----------------------------------------------------------------
 */

typedef struct
{
    int   (*cmp)();     /* compares two objects                */
    int   (*swap)();    /* swap two objects                    */
    int   itemsize;     /* size of one element in heap         */
    int   nitems;       /* Number of items in the heap         */
    int   maxitem;      /* Maximum number of items in heap     */
    char  *bottom;      /* Ptr. to most-recently added item    */
    char  *heap;        /* pointer to the heap itself          */
}
PQ;

/*----------------------------------------------------------------*/

static  void    reheap_down( p )
PQ      *p;
{
    /*  Reheap the Heap, starting at the top and working
     *  down;
     */

    int     parent;     /* index of parent           */
    int     child;      /* index of child            */
    char    *pparent;   /* pointer to parent         */
    char    *pchild;    /* pointer to child          */
    char    *psibling;  /* pointer to child's sibling */
    char    *heap;      /* pointer to heap           */

    heap = p->heap;

    for( parent = 0, child = 1;  child < p->nitems ;)
    {
        pparent = heap + (parent * p->itemsize);
        pchild  = heap + (child  * p->itemsize);

        if( child+1 < p->nitems )
        {
            psibling = pchild + p->itemsize ;

            if( (*p->cmp)(pchild, psibling)  < 0 )
            {
                pchild = psibling;
                ++child;

            }
        }

        if( (*p->cmp)( pparent, pchild ) >= 0)
            break;

        (*p->swap)( pparent, pchild );

        parent = child;
        child  = (parent * 2) + 1;
    }
}

/*----------------------------------------------------------------*/

static void     reheap_up( p )
PQ      *p;
{
    /* Reheap the Heap, starting at the bottom and working up.
     * Note that we must use a divide-by-2 rather than a
     * shift-right in the while loop because -1/2 == 0 but
     * -1 >> 1 == -1.
     */

    int     parent;     /* index of parent      */
    int     child;      /* index of child       */
    char    *pparent;   /* pointer to parent    */
    char    *pchild;    /* pointer to child     */
    char    *heap;      /* pointer to heap      */

    child = p->nitems - 1;
    heap  = p->heap;

    while( (parent = (child-1) / 2)  >= 0 )
    {
        pchild  = heap + (child  * p->itemsize);
        pparent = heap + (parent * p->itemsize);

        if( (*p->cmp)( pparent, pchild ) >= 0)
            break;

        (*p->swap)( pparent, pchild );
        child = parent;
    }
}

/*----------------------------------------------------------------*/

PQ      *pq_create( numele, elesize, cmp, swap, initheap )

int     numele;         /* max # of elements in the queue */
int     elesize;        /* size of one element in byte    */
int     (*cmp)();       /* pointer to comparison function */
int     (*swap)();      /* pointer to swap function       */
char    *initheap;      /* inital heap, NULL to allocate  */
{
    /* Create a priority queue that can hold at most
     * "numele" elements, each of size "elesize". The
     * cmp function is passed two pointers to queue
     * elements and it should behave as follows:
     *
     *      (*cmp)(p1, p2)
     *
     * For descending priority queues [pq_get() returns the
     *                                         largest item].
     *      *p1 <  *p2      return <  0
     *      *p1 == *p2      return == 0
     *      *p1 >  *p2      return >  0
     *
     * For ascending priority queues [pq_get() returns the
     *                                         smallest item].
     *      *p1 <  *p2      return >  0
     *      *p1 == *p2      return == 0
     *      *p1 >  *p2      return <  0
     *
     * If the initheap argument is NULL, an empty heap is
     * created automatically, otherwise initheap must point
     * at an initialized numele-element-long heap.
     */

    PQ      *p, *malloc() ;
    int     heapsize;

    heapsize = numele * elesize ;   /* heap size in bytes */

    if( initheap )
    {
        if( !(p = malloc(sizeof(PQ))) )
            return 0;

        p->heap   = initheap;
        p->bottom = initheap + ((numele - 1) * elesize);
        p->nitems = numele;
    }
    else
    {
        if( !(p = malloc(sizeof(PQ) + heapsize)) )
            return 0;

        p->heap   = (char *)(p + 1);
        p->nitems = 0;
        p->bottom = p->heap - elesize ;
    }

    p->cmp      = cmp;
```

```
    p->swap     = swap;
    p->itemsize = elesize;
    p->maxitem  = numele ;
    return p;
}

/*----------------------------------------------------------------*/

pq_ins( p, item )
PQ      *p;             /* Pointer to priority queue     */
char    *item;          /* Pointer to item to insert     */
{
    /* Insert a new item into the priority queue (provided
     * that space is available.
     *
     * Return the number of empty slots in the queue before
     * the insertion. This number is 0 if the queue is
     * full and nothing is inserted. Algorithm is:
     *
     *  if( space is available in the queue )
     *          increase queue size
     *          copy new item into bottom of queue
     *          reheap from the bottom up.
     */

    int space_avail = p->maxitem - p->nitems;

    if( space_avail > 0 )
    {
        ++( p->nitems );
        memcpy( p->bottom += p->itemsize, &item, p->itemsize );
        reheap_up( p );
    }

    return space_avail ;
}

/*----------------------------------------------------------------*/

int     pq_del( p, target )

PQ      *p;             /* pointer to priority queue             */
char    *target;        /* place to copy current largest item */
{
    /* Copy the largest item in the priority queue to
     * the address held in target, then delete the item.
     *
     * Return the number of items in the queue before the
     * delete. If this number is 0, then nothing was
     * in the queue and *target will not have been
     * modified. Algorithm is:
     *
     *  if( there's something in the queue )          [1]
     *      remember pointer to former first item     [2]
     *      replace the first item with the last one  [3]
     *      shrink the heap by one element            [4]
     *      reheap from the top down                  [5]
     */

    int     slots_in_use;

    if( slots_in_use = p->nitems )                    /* 1 */
    {
        memcpy( target,  p->heap,   p->itemsize );    /* 2 */
        memcpy( p->heap, p->bottom, p->itemsize );    /* 3 */

        --(p->nitems) ;                               /* 4 */
        p->bottom -= p->itemsize;

        reheap_down( p );                             /* 5 */
    }

    return slots_in_use ;
}

/*----------------------------------------------------------------*/

char    *pq_look( queue )
PQ      *queue;
{
        /* Return a pointer to the largest/smallest element
         * in the priority queue but don't dequeue it.
         */

        return queue->heap;
}

/*----------------------------------------------------------------*/

int     pq_numele( queue )
PQ      *queue;
{
        /* Return number of items in queue
         */

        return queue->nitems;
}
```

(continued on next page)

Listing One *(Listing continued, text begins on page 102.)*

```
/*------------------------------------------------------------*/
#ifdef MAIN

int Descending = 1;     /* Change these in Codeview     */
int Makequeue  = 1;     /* to change the tests.         */

cmp( s1, s2 )
char    **s1, **s2;
{
        int rval;
        rval = strcmp( *s1, *s2 );

        if( Descending )
            return rval;
        else
            return   ( rval <  0 )  ? 1 :
                     ( rval >  0 )  ? -1 :
                  /* rval == 0 */     0 ;
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

swap( s1, s2 )
char    **s1, **s2;
{
        char    *tmp;

        tmp = *s1;
        *s1 = *s2;
        *s2 = tmp;
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

printq( p )
PQ      *p;
{
    int i;

    printf("Queue is:\n");

    printf( "\tcmp............. 0x%04x\t", p->cmp      );
    printf( "\tswap............ 0x%04x\n", p->swap     );
    printf( "\titemsize........ %d\t",     p->itemsize );
    printf( "\tnitems.......... %d\n",     p->nitems   );
    printf( "\tmaxitem......... %d\t",     p->maxitem  );
    printf( "\tbottom.......... 0x%04x\n", p->bottom   );
    printf( "\theap............ 0x%04x\t", p->heap     );
    printf( "\tbottom - heap... %d\n\n",    (char **)p->bottom -
                                            (char **)p->heap );
    if( p->nitems <= 0 )
        printf("\tqueue is empty\n");

    else for( i = 0 ; i < p->nitems ; i++ )
    {
       printf("\t%-2d: %10.10s (0x%04x) (children: %2d, %2d)\n",
              i, ((char **)p->heap)[i], ((char **)p->heap)[i],
                 (2*i)+1,                 (2*i)+2               );
    }
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

main()
{
    PQ    *queue;
    char  buf[80];
    int   i;
    char  *p;

    static char  *testq[] =
    {
        "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"
    };

    if( Makequeue )
        queue = pq_create( 10, sizeof(char*), cmp, swap, 0     );
    else
        queue = pq_create( 10, sizeof(char*), cmp, swap, testq);

    if( !queue )
    {
        printf("pq_create failed\n");
        exit( 1 );
    }

    printf("+-----------------------------------------+\n");
    printf("| Enter i<string><CR> to add string to    |\n");
    printf("| queue, d<CR> to dequeue top element, q to |\n");
    printf("| exit the program.                       |\n");
    printf("+-----------------------------------------+\n");

    while( 1 )
    {
        printq( queue                         );
        printf( "\n[i|d|q][string] :"          );
        gets  ( buf                           );

        if( *buf == 'q' )
            exit( 1 );

        else if( *buf == 'i' )
        {
            i = pq_ins( queue, strsave(buf + 1) );

            if( i )
              printf("%d slots avail. before insert\n", i);
            else
              printf("Queue was full, did nothing\n\n");
        }
        else
        {
            i = pq_del( queue, (char*) &p );
            printf("%d slots used before delete, got <%s>\n",
                                                         i, p);
            if( i )
            {
                free( p );
                p = "nothing";
            }
        }
    }
}

#endif
```

End Listing One

Listing Two

```
Listing 2 -- strsave.c
```

```
char    *strsave( str )
char    *str;
{
    /*  Save the indicated string in a malloc()ed section
     *  of static memory. Return a pointer to the copy or
     *  0 if malloc failed.
     */

    register char *rptr;
    extern   char *malloc();

    if( rptr = malloc( strlen(str) +1 ))
    {
          strcpy( rptr, str );
          return rptr;
    }

    return (char *)0;
}
```

End Listing Two

Listing Three

```
Listing 3 -- freq.c
```

```
#include <stdio.h>

/*      FREQ.C          Print a list of the frequency
 *                      of occurance of all bytes
 *      in a list of files given on the command line.
 *      Frequencies are printed as a probability x 100.
 *      For example, if we read a total of 20 characters,
 *      5 of which are 'e', the probability of an 'e'
 *      occuring in the input is 5/20 (.25) and freq will
 *      output (.25 * 100) or 25.
 */

typedef struct
{
        int     val;
        double  count;
} ITEM;

#define TABSIZE 256
ITEM    Tab[ TABSIZE ];   /* I'm counting on this being    */
                          /* initialized to zero.          */

/*--------------------------------------------------------*/

cmp( item1, item2 )
ITEM    *item1, *item2;
{
    /* Comparison function used by ssort(), below.
     * Count is the primary sort field and val is
     * the secondary field.
     */

    int rval;
```

```
    return   ( item1->count <  item2->count )   ? -1 :
             ( item1->count >  item2->count )   ?  1 :
           /* item1->count == item2->count */
                                   item1->val - item2->val ;
}

/*------------------------------------------------------------*/

main( argc, argv )
char    **argv;
{
    char          *bin_to_ascii();      /* in pchar.c */
    FILE          *fp;
    int           i;
    double        smallest;
    double        largest;
    double        numchars    = 0.0 ;
    double        sum         = 0.0 ;
    double        probability = 0.0 ;

    reargv( &argc, &argv );       /* Needed only for      */
                                  /* On Command! shell    */

    for( --argc, ++argv ; --argc >= 0 ; argv++ )
    {
        if( !(fp = fopen( *argv, "rb")) )
            perror( *argv );
        else
        {
            fprintf( stderr, "%s\n", *argv );

            while( (i = getc(fp)) != EOF )
            {
                ++ numchars;
                ++ Tab[ i & 0xff ].count ;
            }

            fclose( fp );
        }
    }

    /* Find largest and smallest elements and at the same
     * time initialize the val fields of the Tab entries.
     */

    largest  = 0.0;
    smallest = numchars ;

    for( i = 0 ; i <= 0xff ; i++ )
    {
        Tab[i].val = i;

        if( Tab[i].count > 0.00000001
        &&  Tab[i].count < smallest )
                smallest = Tab[i].count;

        if( Tab[i].count > largest )
                largest = Tab[i].count;
    }

    /* Sort the list by probability. A shell sort is used.
     * You can replace the ssort call below with a call to
     * the qsort() subroutine, available with many compilers.
     * Qsort() takes the same arguments as ssort().
     * A version of qsort appeared in the in the C Chest,
     * DDJ #102 (April, 1985; also Bound Volume 10, p.316).
     * The ssort() subroutine appeared in DDJ #113, C Chest
     * (March, 1986) p. 70.
     */

    ssort( Tab, TABSIZE, sizeof(ITEM), cmp );

    /*
     * Print the list. Each element is printed as three
     * numbers: the value of the charcter (in hex), the
     * probability of that character apearing in the input,
     * and the probabability normalized so that the least-
     * frequently occuring probability has the value 1.
     */

    for( i = 0 ; i < TABSIZE ; i++ )
    {
            probability = Tab[i].count / numchars ;
            sum         += probability;

            printf( "0x%02x\t%7.6f\t%1.0f\n",
                    Tab[i].val,
                    probability,
                    Tab[i].count / smallest  );
    }

    fprintf( stderr, "Total = %8.6f  (N = %8.2f)\n",
                                    sum, numchars );
}
```

End Listings

Listing One *(Text begins on page 116.)*

```
Listing 1. Inheritance in SCOOPS

; (C) Copyright 1987 Ernest R. Tello

(define-class artifact
                    (instvars material weight purpose cost)
                    (options
                          (gettable-variables material weight purpose cost)
                           settable-variables
                           inittable-variables))

(define-class transport-means
                    (instvars medium time-range power-source)
                    (mixins artifact)
                    (options
                         (gettable-variables medium time-range power-source)
                           settable-variables
                           inittable-variables))

(define-class transport-vehicle
                    (instvars load-capacity length max-speed)
                    (mixins artifact transport-means)
                         (options
                          (gettable-variables load-capacity length max-speed)
                           settable-variables
                           inittable-variables))


(define-class passenger-vehicle
                    (instvars capacity safety dining facilities)
                    (mixins artifact transport-means transport-vehicle)
                    (options
                          (gettable-variables capacity safety dining facilities)
                           settable-variables
                           inittable-variables))

(define-class water-transport-vehicle
                    (classvars ( body-name 'hull) (dof 2) (dangers 'sink ) (advantages  'relaxing ))
                    (mixins artifact transport-means transport-vehicle passenger-vehicle)
                    (options
                          (gettable-variables dof dangers)
                           settable-variables
                           inittable-variables))

(define-class surface-vessel
                    (instvars #-decks #-masts #-engines )
                    (mixins artifact transport-means transport-vehicle passenger-vehicle water-transport-
vehicle)
                    (options
                          (gettable-variables #-decks #-masts #-engines )
                           settable-variables
                           inittable-variables))

(define-class ship
                    (instvars
                          x-position y-position x-velocity y-velocity mass)
                    (mixins surface-vessel)
                  (options
                    (gettable-variables x-position y-position x-velocity y-velocity mass)
            settable-variables
            inittable-variables))

(define-method (ship speed) ()
                         (sqrt (+ (expt x-velocity 2)
                                     (expt y-velocity 2))))

(define-method (ship direction) ()
                         (atan y-velocity x-velocity))

(define-class ocean-liner
                         (instvars company launched homeport tons )
                    (mixins ship)
                    (options
                          (gettable-variables company launched homeport tons )
                           settable-variables
                           inittable-variables))

(define ship1
  (make-instance ship
                           'x-position 100
                           'y-position 150
                           'x-velocity 30
                           'y-velocity 40
                           'mass 100))

(compile-class artifact)

(compile-class transport-means)

(compile-class transport-vehicle)

(compile-class passenger-vehicle)

(compile-class water-transport-vehicle)

(compile-class surface-vessel)

(compile-class ship)
```

```
(compile-class ocean-liner)
```

End Listing One

Listing Two

```
Listing 2. Multiple Inheritance in SCOOPS

; (C) Copyright 1987 Ernest R. Tello

(define-class business
          (instvars name location industry business-type size
                year-founded ownership-type gross-sales costs

 market-share)
           (options
                     (gettable-variables  name location industry
                                 business-type size year-founded
                                                  ownership-type

 gross-sales costs market-share)
           settable-variables
           inittable-variables))

 (define-method (business calc-net-gain) (gross-sales costs)
           (- gross-sales costs))

 (define-class adversary
           (instvars aggressiveness allies goals common-goals
                                       strengths weaknesses)
                     (options
                               (gettable-variables aggressiveness
                                        allies goals common-goals
                                            strengths weaknesses)
                             settable-variables
                             inittable-variables))

 (define-class competitor
                             (mixins business adversary))

 (compile-class business)
 (compile-class adversary)
 (compile-class competitor)

 (define your-business (make-instance business))

 (define competitor-1 (make-instance competitor))
```

End Listings

MICRO/SYSTEMS JOURNAL

FOR THE ADVANCED COMPUTER USER

SUBSCRIBE NOW AND SAVE OVER 15% OFF THE NEWSSTAND PRICE!

$5 SAVINGS

Yes! I want to subscribe to Micro/Systems Journal™ and save over 15% off the cover price.

☐ 1 Year (6 issues) $20 ☐ 2 Years (12 issues) $35

☐ Please charge my: ☐ Visa ☐ MasterCard ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # ______ Exp. date ______

Signature ______

Name ______

Address ______

City ______ State ______ Zip ______

Savings based on a full one-year cover price of $23.70. Canada and Mexico add $3 for surface mail, $7 for airmail per year. All countries add $12 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

3013

A Publication of M & T Publishing, Inc.

$5 SAVINGS

Yes! I want to subscribe to Micro/Systems Journal™ and save over 15% off the cover price.

☐ 1 Year (6 issues) $20 ☐ 2 Years (12 issues) $35

☐ Please charge my: ☐ Visa ☐ MasterCard ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # ______ Exp. date ______

Signature ______

Name ______

Address ______

City ______ State ______ Zip ______

Savings based on a full one-year cover price of $23.70. Canada and Mexico add $3 for surface mail, $7 for airmail per year. All countries add $12 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

3013

A Publication of M & T Publishing, Inc.

$5 SAVINGS

Yes! I want to subscribe to Micro/Systems Journal™ and save over 15% off the cover price.

☐ 1 Year (6 issues) $20 ☐ 2 Years (12 issues) $35

☐ Please charge my: ☐ Visa ☐ MasterCard ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # ______ Exp. date ______

Signature ______

Name ______

Address ______

City ______ State ______ Zip ______

Savings based on a full one-year cover price of $23.70. Canada and Mexico add $3 for surface mail, $7 for airmail per year. All countries add $12 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

3013

A Publication of M & T Publishing, Inc.

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

For the Advanced Computer User
Micro/Systems Journal

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

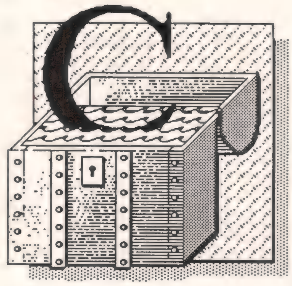
FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

For the Advanced Computer User
Micro/Systems Journal

Box 3713
Escondidio, CA 92025-9843

Priority Queues



This month is the first of what has unintentionally become a two-part article. I had intended to implement an interesting variant of the Huffman encoding data-compression algorithm that is used by the Unix COMPACT program and is described in Robert Gallager's article "Variations on a Theme of Huffman" (*IEEE Transactions on Information Theory*, vol. IT-24, no. 6 [November 1978]: 668–674). Gallager describes an adaptive one-pass Huffman code in which the code changes as the input file is processed. This way the code tree stays optimal over the life of the file. As usual, the problem proved more intractable than I had at first anticipated. So, this month I'll describe some of the stuff I developed along the way to a solution; I'll discuss the actual compression algorithm in a future column.

Priority Queues

A *queue*, in the normal data-structure sense of the word, works like a line in a bank does. New entries are added to the back of the queue, and items are removed from the front. That is, the items in the queue are ordered by insertion time—those items

by Allen Holub

that were inserted first are removed first. Another kind of queue is a priority queue or heap, a queuelike data structure in which something other than time is used to order the elements. For example, the largest or smallest element—rather than the least-recently inserted element—could be the first to be dequeued. Heaps have several uses (other than Huffman codes). For example, a heap is used to do the merge phase of the external-sorting program described in the June 1986 C Chest. A clever sorting algorithm (heapsort) uses a priority queue to do the sorting. Elements are put into the queue in random order, and the smallest element is extracted repetitively until the array is sorted.

The file PQ.C (Listing One, page 94) contains a set of general-purpose priority queue routines. Queues can be constructed of any sort of object (numbers, pointers, structures, and so forth) and can be ordered in any sort of way. You could, for example, keep a queue of structures, adding structures to the queue at random but always extracting the one with the smallest key. You could also create a queue of pointers to structures. You could even use these routines to maintain a normal (but inefficient) queue by adding a time-entered field to the structure and then extracting by smallest time entered. Similarly, a stack can be represented by a priority queue in which the item with the largest time entered is removed first.

PQ.C contains five externally accessible routines. The first of these creates a new queue:

```
typedef char    QUEUE;
QUEUE    *pq_create( numele,
            elesize, cmp, swap,
            initheap )
int numele;
int elesize;
int    (*cmp)( );
int    (*swap)( );
void  *initheap;
```

The *QUEUE* type is a dummy pointer-size type that is used in the same way as is the *FILE* pointer returned from *fopen( )*. *Pq_create( )* is passed five arguments. *Numele* is the maximum number of elements in the queue. *Elesize* is the size of one element. *Cmp* is a pointer to a comparison function. It is passed pointers to two enqueued objects and should return a value as indicated in Table 1, page 105. *Swap* is a pointer to a swap function. It is passed pointers to two objects, and it should swap the objects (not the pointers). Finally, *initheap* can be used in two ways: if it is *NULL*, an empty priority queue is created; otherwise, *initheap* is assumed to be a pointer to an already-initialized *numele*-long array that will be used as the heap. I'll discuss the utility of this in a moment. I've used the ANSI syntax here to declare *initheap*. A *void* pointer is one that doesn't point at any explicit object; you have to cast it to a pointer to a real type to use it. If your compiler doesn't support *void* pointers, use a pointer to *char*.

A queue that's created by *pq_create( )* can be deleted by *free( )*; just pass it the pointer returned from *pq_create( )*. Note, however, that this will only free memory allocated by *pq_create( )* itself. If you create your own initial queue and pass it to *pq_create( )* via the *initqueue* argument, you'll have to free up the memory that you allocated. By the same token, if you create a queue of pointers to strings, *free( )* will free the memory used by the queue but not by the strings.

I'll illustrate how to set up a queue

C CHEST
(*continued from page 102*)

with a moderately complex example—say you want to keep a queue of pointers to the following structures:

```
typedef struct
{
  int weight;
  char *stuff;
  long more_stuff;
{
ITEM;
```

The *ITEMs* can be inserted in any order, but they will be extracted in order of decreasing weight. The comparison function used for this purpose looks like this:

```
cmp( p1, p2 )
ITEM **p1, **p2;
{
  return (*p1)->weight
    - (*p2)->weight;
}
```

and the swap function looks like this:

```
swap( p1, p2 )
ITEM **p1, **p2;
{
  ITEM *tmp;
  tmp = *p1;
  *p1 = *p2;
  *p2 = tmp;
}
```

If you had wanted to order the queue by increasing—rather than decreasing—weight, you would have reversed *p1* and *p2* in the comparison function's *return* statement.

An empty ten-element queue can now be created with the following:

```
QUEUE *qp;
qp = pq_create( 10, sizeof(ITEM*),
                        cmp, swap, 0 );
```

Items are added to the queue using:

```
int pq_ins( qp, item )
QUEUE *qp;
char *item;
```

where *qp* is a pointer returned from a previous *pq_create( )* call and *item* is a pointer to the object to insert. *Pq_ins( )* returns the number of empty slots that were in the queue

before the insert. If the return value is 0, the queue was full and *pq_ins( )* will have done nothing. A new item is inserted in the queue that I created earlier with the following code:

```
ITEM *p;
p = (ITEM *) malloc (sizeof(ITEM));
p->stuff = "A string";
p->weight = 5;
if( !pq_ins( qp, &p ) )
    printf("queue is full");
```

Note that you have to pass in the address of the object to enqueue—in this case the address of the pointer *p*. Items are extracted from the queue with:

```
int pq_del( qp, item )
QUEUE *qp;
char *item;
```

where, again, *qp* is a *QUEUE* pointer returned from *pq_create( )* and *item* is a pointer to a place into which the dequeued object will be copied. The number of items in the queue before the delete is returned. If this number is 0, the queue was empty and the contents of **item* are undefined. For example, the largest element of the queue can be dequeued with the following code:

```
ITEM *p;
if( !pq_del( qp, &p ) )
printf("queue is empty");
```

Two additional support routines are provided:

```
char *pq_look( p )
char *pq_numele( p )
QUEUE *p;
```

Again, *p* is a pointer to a *QUEUE* returned from a previous *pq_create( )* call. *Pq_look( )* returns a pointer to the object at the head of the queue. The object is not actually dequeued, however. *Pq_numele( )* returns the number of elements currently in the queue.

Implementation

A priority queue can be represented by a binary tree that has the following properties:

1. All children in the tree have a value less than their parent.
2. The tree is as perfectly balanced as possible—that is, the difference in height of all leaves is at most 1.
3. Leaves are inserted into the tree from left to right until an entire rank is full, then the next rank is started.

An example tree is shown in Figure 1, below. Note that this tree is not strictly ordered in the normal way. That is, rule 1 doesn't require that the tree be sorted, only that the children are smaller than the parent. You could exchange the subtrees rooted at 1 and 2 without violating rule 1. This ordering guarantees that the root node of the entire tree always holds the largest element, however. Notice that an insert operation is harder than normal because of rules 2 and 3. An 11th node must be inserted as the right child of node 4 and a 12th node as the left child of node 5. If you do this, however, rule 1 may be violated by the newly inserted node. To avoid this last problem, you have to adjust the contents of the nodes at every insertion (a process called reheaping). For example, if you insert a new node having the value *k* as the right child of node 4, you'll have to shuffle things around because of rule 1.

The reheaping process is illustrated in Figure 2, page 106. You reheap from the bottom up with an insertion. Because *k* is greater than *h*, the contents of nodes 10 and 4 must be swapped. You then go up one level to node 4 and compare its key to its parent's. Here *k* is still greater than *i*, so you swap again, moving the *k* to node 1. Finally, nodes 0 and 1 must be swapped as well, moving *k* to the root position.

Because of rules 2 and 3, it's convenient to represent the tree as an array in which the element at node *N* has children at nodes $2N+1$ and $2N+2$. For example, the node at *array[0]* has children at *array[1]* and *array[2]*, the node at *array[2]* has children at *array[5]* and *array[6]*, and so forth. This array representation, usually called a heap, has both advantages and disadvantages. The main

For an ascending queue (the smallest object is dequeued first): if:	`(*cmp)(p1,p2)` returns:	For a descending queue (the largest object is dequeued first): if:	`(*cmp)(p1,p2)` returns:
`*p1 > *p2`	negative number	`*p1 > *p2`	positive number
`*p1 == *p2`	0	`*p1 == *p2`	0
`*p1 < *p2`	positive number	`*p1 < *p2`	negative number

Table 1: *Values returned from the comparison function*

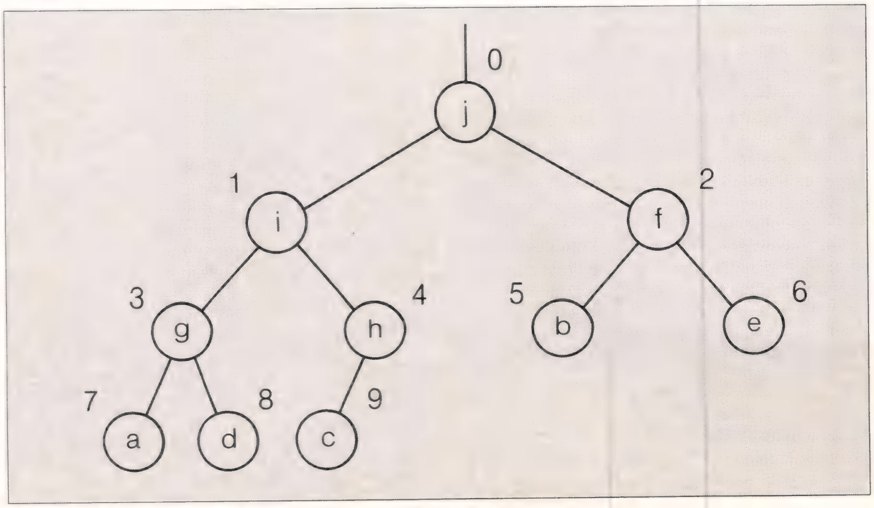


Figure 1: *A priority queue represented as a binary tree*

(continued from page 105)

problems are that the maximum number of elements must be known at create time, and it's difficult to delete an interior node at random or to merge two queues. Nonetheless, the heap representation is the most efficient for the overwhelming majority of applications—in which the only operations are insert and delete.

The tree from Figure 1 is shown as an array in Figure 3, right. To insert a node in the tree, you make the array one element larger, put the new node in the rightmost element, and then reheap from the bottom up (right to left). The largest node is always at *array[0]*, just as it was always at the root position in the tree. Delete the largest element by copying the rightmost node in the tree (*array[9]* in Figure 3) into *array[0]*, making the array one element smaller, and then reheaping from the top down (left to right).

The heap representation is a reasonably efficient one. Unlike a normal binary tree, you never have to search for the insertion point in the tree (it's always at the far right). Similarly, the largest element is always in a fixed place (at the far left). Though there's a certain amount of copying that has to be done during a reheap, no more than $\log_2 N$ copies need ever be done. Nonetheless, it's worthwhile to make the queue elements themselves small. Use an array of pointers to structures rather than an array of structures. This way you have to swap only two pointers—rather than two complete structures—when you reheap.

Note that a sorted array is a valid heap (though a heap is not necessarily a sorted array). This property explains the *initheap* argument to *pq_create( )*. If your input is already a sorted list, there's no point in doing a series of insert operations to create the heap—you can just pass the array directly to *pq_create( )*.

The priority queue is represented internally by the following structure:

```
typedef struct
{
   int (*cmp)( );
   int (*swap)( );
```

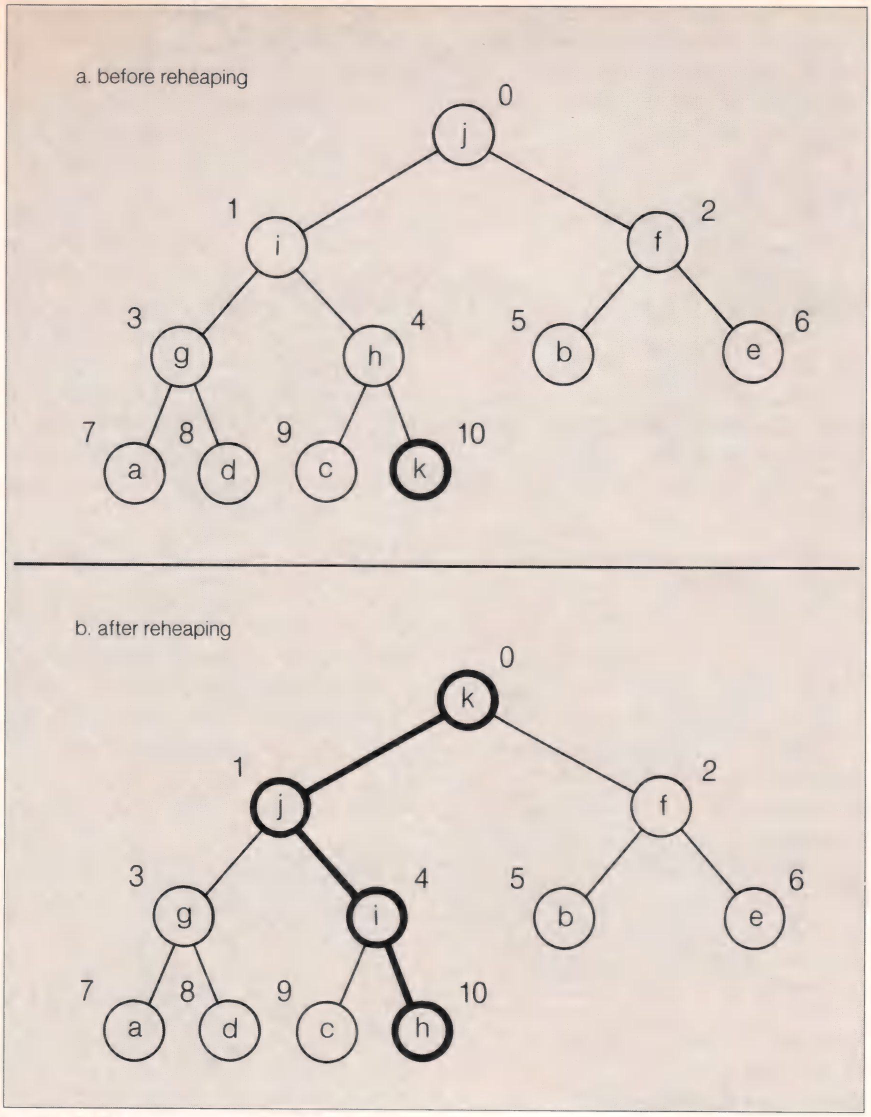


Figure 2: *Inserting a new node*

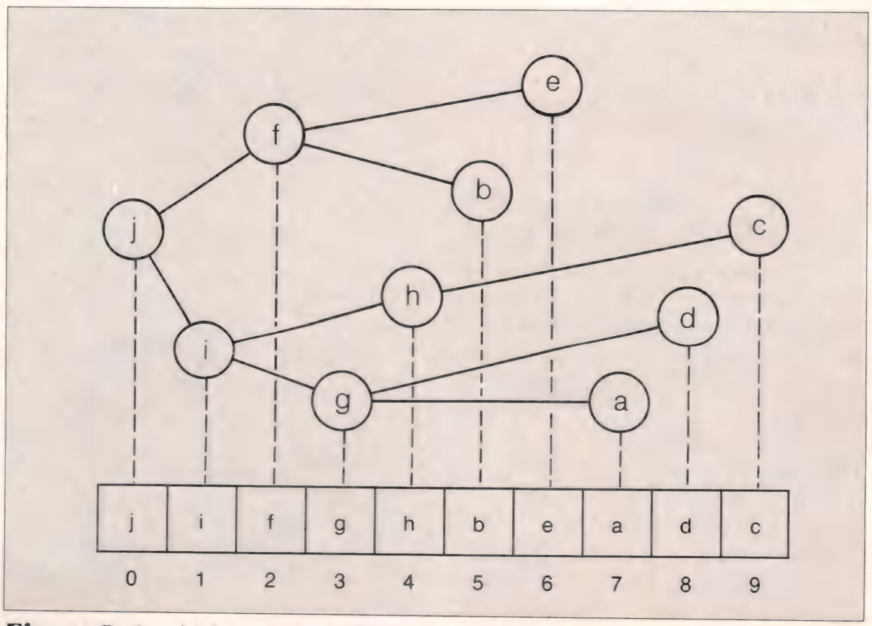


Figure 3: *A priority queue represented as an array*

(continued from page 106)

```
    int itemsize;
    int nitems;
    int maxitem;
    char *bottom;
    char *heap;
{
PQ;
```

The comparison and swap function pointers are remembered in *cmp* and *swap*. *Itemsize* is the size in bytes of one element of the heap, *nitems* is the number of elements currently in the heap, and *maxitem* is the maximum number of items that the heap can hold. The *heap* field points at the array itself, and *bottom* points at the most recently added element. It will point at *heap[-1]* if the heap is empty. Note that you could derive *nitems* by subtracting *bottom* from *heap* but it's more convenient to keep it as a separate number.

The *pq_create( )* function on lines 128–188 of Listing One creates a *PQ* structure and initializes the various fields. If *initheap* is not *NULL*, space for the *PQ* structure alone is allocated (on line 167) and *heap* is made to point to the specified array. The various other fields are adjusted so that the heap is full. If *initheap* is *NULL*, space for both the *PQ* structure and the heap itself is allocated (on line 176), and an empty heap is initialized. Finally, a pointer to the *PQ* structure is returned on line 188.

The insert and delete operations are performed in *pq_ins( )* (lines 193–219) and *pq_del( )* (lines 224–258) using the method described earlier. Because the compiler doesn't know the size of a heap element at compile time, items must be copied with *memcpy( )* calls at run time. The reheaping is deferred to two static workhorse functions—*reheap_down( )* and *reheap_up( )*—declared at the top of the file. *Reheap_down( )* (lines 51–89) starts at the root node (*heap[0]*) and works down the tree. It selects the larger of the two children on lines 72–81 and then swaps the root element and the larger child (and moves down the tree) if necessary. The reheap process stops as soon as the root is larger than both

Flotsam and Jetsam

Standard #include Files

At present I'm using the Microsoft C compiler, Version 4.0, for two reasons—the first is the CodeView debugger, and the second is the degree of Unix compatibility. (The MS-DOS compiler is the Xenix compiler; there's literally no difference.) I spend a lot of time bouncing around between the Unix system at Berkeley and my own PC, so having compatible compilers is essential. Using the Microsoft compiler, I've never had to modify a program written under one system and ported to the other unless that program did some sort of very-low-level communication with the operating system (such as talk directly to the BIOS).

There's another advantage to Unix compatibility, and that's standardization. Although the emerging ANSI standard finally incorporates the I/O library into the language—at least in terms of regularizing the function names and argument-passing conventions—there aren't nearly as many ANSI functions as there are Unix functions, so Unix must continue to provide a de facto standard. You don't really know C unless you know the Unix library. Your code just won't be portable because you don't know what's standard and what isn't. Learning your own compiler's library is a necessary but not sufficient condition for knowing C. As a consequence, it's very much to your advantage to pick up a copy of the Unix System V documentation and read through it. Hitherto, these manuals have been hard to come by. All that you could get were the now-out-of-date Version 7 manuals (the big blue and green paperbacks). AT&T, however, has just published the manuals for Release 2.0 of Unix System V. The complete five-volume set is overkill unless you're really using Unix. Nonetheless, Volume 2—*System Calls and Library Routines*—should be in every C programmer's library. It's available by mail order (for $29.95) from the Computer Literacy bookshop in San Jose, California, and is well worth the money. (Computer Literacy's phone number is [408] 435-1118. The five-volume set is Steven V. Earhart's *Unix Programmer's Manual* [New York: Holt, Rinehart & Winston, 1986].)

Because I develop all the programs that appear in C Chest in the Unix environment (the Microsoft environment is the Unix environment for all intents and purposes), it seems worthwhile to list the various *#include* files that I use in C Chest. These are usually presented without comment because they're both standard and well documented elsewhere. If your compiler doesn't have one of these files, then you don't need to *#include* it in your program because none of the library functions will require any of the things included within the file. By the same token, if your compiler doesn't have one of these files, then it's not Unix compatible, in spite of what the advertisements may say.

It's impossible for me to describe how to port programs to every nonstandard compiler on the market. You'll have to learn how to do this yourself. You'll need to know your own compiler's library pretty well and have at least a working knowledge of the Unix equivalents to various library functions. Do your homework. I will say, while on the subject, that the Lattice compiler is one of those that falsely claims Unix compatibility—for example, it doesn't support a Unix-compatible *stat( )* function, it uses the name *fork( )* in incorrect ways, and it doesn't have a Unix-compatible *#include* file system. These inconsistencies are surprising because Lattice seems to have gone to a lot of trouble to add Unix-compatible functions to its library in the last release. Unfortunately, the degree of Unix compatibility that is indeed present lures you into a false sense of security.

The various *#include* files that you're likely to see in a C Chest pro-

children. Note that an ascending queue (one where the root holds the smallest rather than the largest element) can be created by modifying the comparison function, without touching the reheap code at all.

Reheap_up( ) (lines 95-124) reheaps in the other direction. It starts with the most recently entered item (which is at *heap[nitems-1]*) and works up the tree to the root. The routine is simpler than *reheap_down( )* because you don't have to find the larger sibling—a child has only one parent. Again, the reheaping processes stops as soon as a parent node that is larger than the current child node is detected.

The remainder of the file is compiled only if *MAIN* is *#defined*. In this case, a stand-alone test program is compiled. This program creates a ten-element-long heap of character pointers and then adds or deletes strings from the heap according to commands entered from the keyboard. Type *i<string>* to insert *<string>* into the heap, *d* to delete an item, and *q* to exit from the program. The priority queue is created either on lines 359 or 361, depending on the value of *Makequeue*. If *Makequeue* is set, an empty queue is created; otherwise, a preinitialized queue (using the array declared on lines 353-356) is created. If *Ascending* is true, the queue is an ascending priority queue (items will be removed in ascending order); otherwise, it's descending. There's no explicit code to modify *Makequeue* and *Ascending* (I just modified them with CodeView, the Microsoft debugger, as I was debugging).

New items are inserted into the queue on line 386 and deleted on line 395. The comparison and swap functions are declared on lines 289-313. Finally, the *strsave( )* function, used on line 389, is shown in Listing Two, page 96. Because this is just a small test program, I'm ignoring the error return from *strsave( )*—you shouldn't do this in a real application, of course.

FREQ.C

The priority queue routines are of general utility. You need a few special-purpose utilities to make Huff-

gram are listed below. Most of these are both Unix and Microsoft compatible, but some are used just in the DOS compiler. Note that this isn't a list of all the Unix/Microsoft *#include* files—just the ones I'm likely to use.

ctype.h—contains various text processing macros: *isalpha*, *isupper*, *islower*, *isdigit*, *isxdigit*, *isspace*, *ispunct*, *isalnum*, *isprint*, *isgraph*, *iscntrl*, *isascii*, *toupper*, *tolower*, and *toascii*.

dos.h—not Unix compatible. Contains *#defines* for the MS-DOS interface functions: *bdos*, *dosexterr*, *int86*, *int86x*, *intdos*, and *segread*.

errno.h—*#defines* for the various error condition codes returned from the I/O library: *EZERO*, *EPERM*, *ENOENT*, *ESRCH*, *EINTR*, *EIO*, *ENXIO*, *E2BIG*, *ENOEXEC*, *EBADF*, *ECHILD*, *EAGAIN*, *ENOMEM*, *EACCES*, *EFAULT*, *ENOTBLK*, *EBUSY*, *EEXIST*, *EXDEV*, *ENODEV*, *ENOTDIR*, *EISDIR*, *EINVAL*, *ENFILE*, *EMFILE*, *ENOTTY*, *ETXTBSY*, *EFBIG*, *ENOSPC*, *ESPIPE*, *EROFS*, *EMLINK*, *EPIPE*, *EDOM*, *ERANGE*, *EUCLEAN*, and *EDEADLOCK*.

fcntl.h—contains definitions needed to use the unbuffered I/O function *open( )*: *O_RDONLY*, *O_WRONLY*, *O_RDWR*, *O_APPEND*, *O_CREAT*, *O_TRUNC*, *O_EXCL*, *O_TEXT*, and *O_BINARY*.

io.h—contains function declarations for *access*, *chmod*, *chsize*, *close*, *creat*, *dup*, *dup2*, *eof*, *filelength*, *isatty*, *locking*, *lseek*, *mktemp*, *open*, *read*, *rename*, *setmode*, *sopen*, *tell*, *umask*, *unlink*, and *write*.

math.h—contains definitions for *abs*, *acos*, *asin*, *atan*, *atan2*, *atof*, *bessel*, *cabs*, *ceil*, *cos*, *cosh*, *exp*, *fabs*, *floor*, *fmod*, *frexp*, *hypot*, *labs*, *ldexp*, *log*, *log10*, *matherr*, *modf*, *pow*, *sin*, *sinh*, *sqrt*, *tan*, and *tanh*.

process.h—contains definitions for various process-control functions: *abort*, *execl*, *execle*, *execlp*, *execlpe*, *execv*, *execve*, *execvp*, *execvpe*, *exit*, *_exit*, *getpid*, *spawnl*, *spawnle*, *spawnlp*, *spawnlpe*, *spawnv*, *spawnve*, *spawnvp*, *spawnvpe*, and *system*.

signal.h—contains definitions needed by the *signal( )* subroutine. Some common definitions are: *SIGINT*, *SIGFPE*, *SIG_DFL*, and *SIG_IGN*.

stdarg.h—contains definitions needed to write an ANSI-compatible subroutine with a variable number of arguments (see varargs.h). The following are defined: *va_list*, *va_start*, *va_arg*, and *va_end*.

stdio.h—contains *#defines* and so on for all the buffered I/O functions (*fopen*, *fprintf*, and so forth). *Stdin*, *stdout*, *stderr*, *FILE*, *EOF*, and *NULL* are all defined here. Note that several psuedofunctions that you're used to thinking of as subroutines (*getchar*, *putchar*, *getc*, *putc*, *feof*, *ferror*, and *fileno*) are actually macros that are *#defined* in stdio.h. If you forget to *#include* this file, you'll get error messages from the linker (such as "Unresolved external:_putchar"). *Putchar* is a macro that ultimately evaluates to a call to a system-level subroutine usually called either *_flsbuf* or *_flushbuf*.

sys/stat.h—contains definitions for the system-status subroutines *stat( )* and *fstat( )*.

sys/types.h—also required by *stat( )* and *fstat( )*. Various time functions (such as *utime( )*) use the information declared here as well.

varargs.h—Definitions for the variable-number-of-argument mechanism used by Unix System V (see stdarg.h). The following are defined here: *va_list*, *va_dcl*, *va_list*, *va_alist*, *va_start*, *va_arg*, and *va_end*.

(continued from page 109)

man trees, however. One of these is the FREQ.C program shown in Listing Three, page 96. FREQ.EXE is a standalone program that takes as input a list of files and outputs a table showing the frequency of occurrence of every 8-bit pattern in these files (I'll call these patterns "characters" from here on). The output table is sorted in ascending order of frequency (least frequently occurring characters toward the top). The output format uses one character per line with three numbers on each line—the leftmost number is the value of the character itself (in hex); the next two numbers are character frequencies, output in two forms. The first form is the probability of occurrence of every character—the number of times that that character occurred in the input divided by the total number of input characters. The second form is a normalized character count in which the least frequently occurring nonzero pattern has the value of 1—it's the character count divided by the count associated with the least frequently occurring character.

The table itself is declared on lines 13–21. It is a 256-element array of *ITEM*s, indexed by character value. One field of the *ITEM* structure holds a count that is incremented every time the associated character is encountered in the input. The other field holds the character value itself. You can derive this from the index, of course, but putting it into an *ITEM* lets you sort the array by frequency of occurrence without loosing the value.

The array is loaded with the *for* loop on lines 58–74. The count associated with every byte is incremented on line 69. The counts are all initialized to 0 by default because global-level objects are always initialized to 0 unless there's an explicit initializer present as part of the declaration. The next *for* loop, on lines 83–93, initializes the *value* fields of the array and at the same time finds the element having the smallest nonzero value. (I'm getting the largest value, too, but am not using it for anything at present.) The array is sorted with the *ssort( )* call on line 105. This subroutine works just like the Unix-compatible *qsort( )* does, but it does a shell sort. See the comment on lines 95–102 for more information. The comparison function used by *ssort( )* is declared on lines 25–39. Finally, the table is printed (and the probabilities and so forth are computed) in the loop on lines 115–126. The sum of the probabilities is printed to *stderr* on line 126 just to make sure that everything worked correctly. It should always be 1.00.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

Bibliography

Fredman, Sedgewick, Sleator, and Tarjan. "The Pairing Heap: A New Form of Self-Adjusting Heap." *Algorithmica* 1:1 New York: Springer Verlag, 1986: 111–129. This describes a priority queue represented as a tree (in order to make the merge and random-delete operations more efficient).

Holub, Allen. "C Chest." *DDJ* (June 1986): 26–40. This article describes a set of general-purpose queue-manipulation routines. The same article appears in *DDJ Bound Volume 10*: 436–334.

Sedgewick, Robert. *Algorithms*. Reading, Mass.: Addison-Wesley, 1983: 127–141. This book contains a description of priority queues (or heaps) and heapsort.

DDJ

(Softstrips begin on page 92.)

(Listings begin on page 94.)

Vote for your favorite feature/article.
Circle Reader Service **No. 8**.

Ever Program On A Silver Platter??

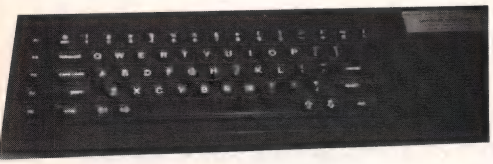
How much would you expect to pay for a 32 bit MC 68000 computer that's a mainframe condensed down into a keyboard? How about $389.00!!?? If it makes you feel any better simply add a zero to the price when you order! But that's actually our price!!! The most powerful computer money can ever buy is now the most inexpensive computer money can buy!!! So don't buy the name! Buy the power!! The power is **not** in the name!

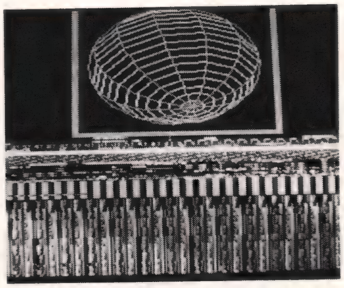
If **you** had the opportunity to work amongst Machine Code ROM Designers, VAX & UNIX wizards in a research laboratory, designing an MC 68000 based computer that's 2nd to none. . .

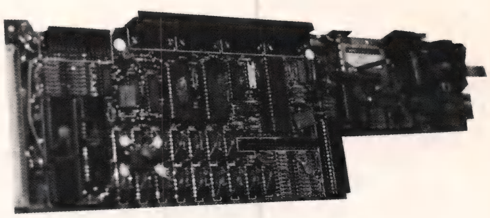
What would you come up with?? And what would you call it??

Well It's Already Been Done!!
They Called It The QL For The Quantum Leap It Is!!

Absolutely a Quantum Leap beyond what you know & use – and it's truly like Programming on a Silver Platter!!







The QL Desktop Minicomputer: Designed by SRL Labs, manufactured by Samsung. An absolute Quantum Leap beyond all the rest! The phenomenal open architecture QDOS: with Virtual Memory RAM, Multitasking Job Control, Multiuser Networking. It'll Cache Files into unused Memory and create/delete Directories Automatically! Even allows File Names up to 36 characters long! Everything is built into ROM here: QDOS, Networking, Windowing, & 32 Bit SuperBasic, all in a totally concurrent non-destructive environment. Unlimited quantities & lengths allowed with: Variables, Program Lines, CONsoles & Buffers. Dynamic non-destructive virtual RAM Disking & Networking buffers too! Even a System Variables Brain Page Screen! Built-in DCE & DTE Serial Ports.

Language Environments:
Metacomco's "C", LISP, BCPL, 68000 Assembler, APL, Development Kits. Prospero's Pro Pascal & Pro Fortran 77. Digital Precision's Forth-83. QJUMP's 65C02 or 8088 Cross Assembly ROMs. Everything generates native 68000 Compiled Code. ROM Firmware & Software Package is now available which will even bring it up in CPM!

Imagine working with a 32 bit SuperBasic that's structured like Turbo Pascal, powered beyond PIC Basic, in an interpretor always present with QDOS, all concurrently running in a built-in UNIX-like multitasking job controlled environment with access to 360 fully channeled windows, devices & files by EACH job! 3 Major Compilers already exist for the SuperBasic source alone! TURBO, SUPERCHARGE, QLIBERATOR! The compiled SuperBasic code or ANY other language will multitask and control with QDOS and SuperBasic. The list of ALL the Superior Features would fill this entire publication!

The QL comes bundled WITH PSION Integrated Word Processor, Spreadsheet, Database and Presentation Graphics Programs. PLUS: Our FREEWARE Demos & Utilities with all purchases! Plus $12 Shipping and Handling

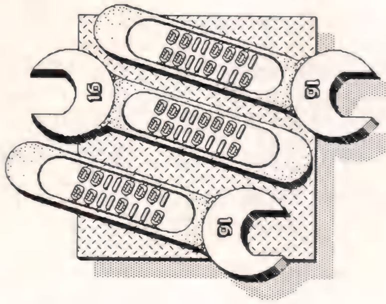
Call: (201) 328–8846

QLine BBS: 328–2919

Technical Info & Assistance – –
Telex: 9102500026
Compuserve ID # 76625,2214

Quantum Computing, Box 1280, Dover, NJ 07801

CIRCLE 144 ON READER SERVICE CARD



Resources

Ward, Robert. *Debugging C*. Indianapolis, Ind.: Que Corp., 1986. 350 pages with index. ISBN 0-88022-261-1.

Finding this book among the flotsam and jetsam of computer publishing is like finding a $100 bill in the street. The art, craft, and/or skill of effective debugging is a topic that has rarely been discussed in any useful way in the computer literature. The debugging chapters in manuals or textbooks are usually focused on defensive strategies and the design of test data, rather than on the stabilization, isolation, localization, and correction of bugs once detected. To new programmers, who have not yet developed a methodical approach to debugging, the process seems a little magical and their own approach is frequently haphazard and based largely on luck. Although native talent is an element of effective debugging, as it is of effective programming, a far larger component is simply the accumulated experience and the strategies learned by finding and fixing many different types of bugs in the past.

In his introduction, Mr. Ward writes: "Until I started teaching, I assumed that good debugging skills were a natural outgrowth of good design skills. Not so. A bright student may intuitively decompose a problem into beautifully coherent, cohesive, functional modules. That same student may not be able to find the

by Ray Duncan

most trivial syntax errors, let alone discover subtle runtime bugs. Equally bright students turn in working designs that literally defy analysis. While I don't believe that we learn debugging by studying design, I do believe that we can learn efficient debugging.

"We can develop a methodological model that directs our efforts toward more productive searches. We can acquire heuristic knowledge (a kind of folk wisdom) about where to look first. We can be deliberately sensitive to the different variables and observable phenomena in different environments. We can become expert at selecting and using appropriate tools. And, through critical analysis of our attempts to find 'worthy' bugs, we can learn from our own mistakes."

The first chapters of *Debugging C* concentrate on the debugging process itself, with emphasis on recognition of bugs (lexical, syntactic, execution, and intent errors), their localization (using the principles of lexical, temporal, or referential proximity), a methodical approach, and good record keeping. Later chapters discuss the localization of compile-time errors and tracing methods. The last chapters of the book are particularly C-oriented, with excellent discussions of the special problems of C programmers: bugs due to data type mismatches, operator precedence or misuse, and uninitialized or out-of-range pointers. Here is where the author gently presents many useful tips that are common sense to veteran C coders, usually acquired through painful experience.

Example: "C, unlike Pascal and Basic, doesn't give an initial value to local variables when it creates them . . . . Most stack frames are relatively small (less than 20 bytes) but every stack frame has a frame pointer (an address higher in the stack) and a return address (an address in the code area). Because uninitialized pointers will use these obsolete addresses which occur frequently in reused areas of the stack, the programmer can expect 10 to 30 percent of the uninitialized pointers to reference code or areas of the stack." Similarly, Mr. Ward explains, with stack traces and memory dumps, why common errors such as initializing too many bytes of an automatic character array result in bizarre transfers of control or crashes "between lines" of source code.

The last few chapters describe the use of some common machine-level and symbolic debuggers, interpreters, and integrated development environments. The book's appendices include the C source code for a debugging subsystem that can be linked to a C application. The debugger provides tracing at several layers of granularity, stack frame displays, memory dumps, and watch points (periodic checks to see if a certain variable has been altered or has taken on a specified value).

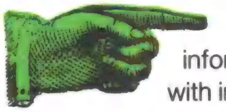
The author couples a direct, lucid, and informal style of writing with a depth of understanding and highly structured presentation that are unusually effective. In spite of the title, most of the book has broad applicability and would be perfectly comprehensible to anyone with even a passing acquaintance with the C language.

Hyman, Michael I. *Memory Resident Utilities, Interrupts, and Disk Management with MS and PC DOS*. Portland, Oreg.: Management Information Source, 1986. 373 pages with index. ISBN 0-943518-73-3.

This book, with the short catchy title, is anything but self-effacing. The introduction reads: "You will find this book to be the ultimate reference guide to getting the most out of your machine. As you read it, you'll learn how to use and enhance DOS to ex-

For Free Info ...

Start Here



Smart buyers start with *DDJ*'s free information card, a shopping center filled with information about the products and services advertised in this very issue: everything from software and systems to peripherals and professional support services.

And smart buyers can use this free information card to quickly and easily gather a comprehensive file of facts, figures and product specs to sort out competing claims. Using *DDJ*'s free information card can prevent you from making the wrong, costly buying decision.

Be a smart shopper. Complete and mail this postage paid card today!

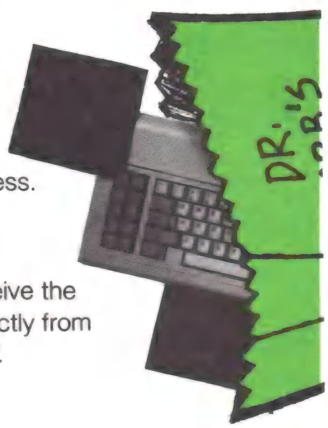
Take a Reader Service Card with You

It's Easy as ...

1. Circle the appropriate free information numbers, referring to the advertiser index for more information.

2. Fill in your name and address.

3. Mail today—postage is absolutely free. You'll receive the product information you need directly from the manufacturers, thanks to *DDJ*.



NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #217, CLINTON, IOWA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

P.O. Box 2157
Clinton, Iowa 52735-2157

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #217, CLINTON, IOWA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

P.O. Box 2157
Clinton, Iowa 52735-2157

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Use this card for FREE, FAST information about the products and services listed in this issue. Simply circle the appropriate numbers below.

1	26	51	76	101	126	151	176	201	226	251	276	301	326	351	376
2	27	52	77	102	127	152	177	202	227	252	277	302	327	352	377
3	28	53	78	103	128	153	178	203	228	253	278	303	328	353	378
4	29	54	79	104	129	154	179	204	229	254	279	304	329	354	379
5	30	55	80	105	130	155	180	205	230	255	280	305	330	355	380
6	31	56	81	106	131	156	181	206	231	256	281	306	331	356	381
7	32	57	82	107	132	157	182	207	232	257	282	307	332	357	382
8	33	58	83	108	133	158	183	208	233	258	283	308	333	358	383
9	34	59	84	109	134	159	184	209	234	259	284	309	334	359	384
10	35	60	85	110	135	160	185	210	235	260	285	310	335	360	385
11	36	61	86	111	136	161	186	211	236	261	286	311	336	361	386
12	37	62	87	112	137	162	187	212	237	262	287	312	337	362	387
13	38	63	88	113	138	163	188	213	238	263	288	313	338	363	388
14	39	64	89	114	139	164	189	214	239	264	289	314	339	364	389
15	40	65	90	115	140	165	190	215	240	265	290	315	340	365	390
16	41	66	91	116	141	166	191	216	241	266	291	316	341	366	391
17	42	67	92	117	142	167	192	217	242	267	292	317	342	367	392
18	43	68	93	118	143	168	193	218	243	268	293	318	343	368	393
19	44	69	94	119	144	169	194	219	244	269	294	319	344	369	394
20	45	70	95	120	145	170	195	220	245	270	295	320	345	370	395
21	46	71	96	121	146	171	196	221	246	271	296	321	346	371	396
22	47	72	97	122	147	172	197	222	247	272	297	322	347	372	397
23	48	73	98	123	148	173	198	223	248	273	298	323	348	373	398
24	49	74	99	124	149	174	199	224	249	274	299	324	349	374	399
25	50	75	100	125	150	175	200	225	250	275	300	325	350	375	400

June '87: Use before September 30, 1987

Name ______________________________
Title ______________________________
Company ____________________ Phone __________
Address ______________________________
City/State/Zip ______________________________

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Use this card for FREE, FAST information about the products and services listed in this issue. Simply circle the appropriate numbers below.

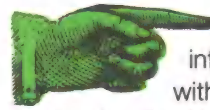
1	26	51	76	101	126	151	176	201	226	251	276	301	326	351	376
2	27	52	77	102	127	152	177	202	227	252	277	302	327	352	377
3	28	53	78	103	128	153	178	203	228	253	278	303	328	353	378
4	29	54	79	104	129	154	179	204	229	254	279	304	329	354	379
5	30	55	80	105	130	155	180	205	230	255	280	305	330	355	380
6	31	56	81	106	131	156	181	206	231	256	281	306	331	356	381
7	32	57	82	107	132	157	182	207	232	257	282	307	332	357	382
8	33	58	83	108	133	158	183	208	233	258	283	308	333	358	383
9	34	59	84	109	134	159	184	209	234	259	284	309	334	359	384
10	35	60	85	110	135	160	185	210	235	260	285	310	335	360	385
11	36	61	86	111	136	161	186	211	236	261	286	311	336	361	386
12	37	62	87	112	137	162	187	212	237	262	287	312	337	362	387
13	38	63	88	113	138	163	188	213	238	263	288	313	338	363	388
14	39	64	89	114	139	164	189	214	239	264	289	314	339	364	389
15	40	65	90	115	140	165	190	215	240	265	290	315	340	365	390
16	41	66	91	116	141	166	191	216	241	266	291	316	341	366	391
17	42	67	92	117	142	167	192	217	242	267	292	317	342	367	392
18	43	68	93	118	143	168	193	218	243	268	293	318	343	368	393
19	44	69	94	119	144	169	194	219	244	269	294	319	344	369	394
20	45	70	95	120	145	170	195	220	245	270	295	320	345	370	395
21	46	71	96	121	146	171	196	221	246	271	296	321	346	371	396
22	47	72	97	122	147	172	197	222	247	272	297	322	347	372	397
23	48	73	98	123	148	173	198	223	248	273	298	323	348	373	398
24	49	74	99	124	149	174	199	224	249	274	299	324	349	374	399
25	50	75	100	125	150	175	200	225	250	275	300	325	350	375	400

June '87: Use before September 30, 1987

Name ______________________________
Title ______________________________
Company ____________________ Phone __________
Address ______________________________
City/State/Zip ______________________________

For Free Info ...

Start Here



Smart buyers start with *DDJ*'s free information card, a shopping center filled with information about the products and services advertised in this very issue: everything from software and systems to peripherals and professional support services.

And smart buyers can use this free information card to quickly and easily gather a comprehensive file of facts, figures and product specs to sort out competing claims. Using *DDJ*'s free information card can prevent you from making the wrong, costly buying decision.

Be a smart shopper. Complete and mail this postage paid card today!

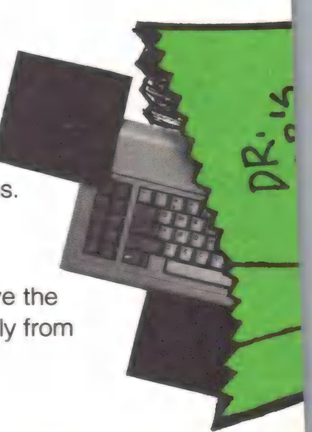
Take a Reader Service Card with You

It's Easy as ...

1. Circle the appropriate free information numbers, referring to the advertiser index for more information.

2. Fill in your name and address.

3. Mail today—postage is absolutely free. You'll receive the product information you need directly from the manufacturers, thanks to *DDJ*.



The Advertiser Index

Advertiser Name	Page No.	Circle No.
Aker Corporation	31	**369**
Aldebaran Laboratories	13	**350**
Alpha Computer Service	81	**321**
Aptech Systems, Inc.	64	*
Austin Code Works	107	**250**
BC Associates	87	**182**
Blaise Computing	2	**159**
Blaise Computing	70	**217**
Borland International	1	**161**
Boston Software Works (The)	30	**384**
Bryte Computer	78	**387**
Burton Systems Software	72	**212**
C Users Group	72	**181**
C Toolbox	126	*
Compu View	125	**122**
CompuServe	65	**237**
Cosmos, Inc.	26-27	**400**
Creative Programming	91	*
Custom Software Systems	104	**268**
DDJ Subscriptions	36	*
Datalight	9	**203**
Desktop A.I.	78	**258**
Digitalk	130	**127**
Ecosoft, Inc.	89	**89**
Essential Software	C3	**138**
Fair-Com	44	**93**
Flexus	95	**189**
Genesis Data Systems	79	**373**
Gold Hill Computers, Inc.	20-21	*
Greenleaf Software	48	**97**
Guidelines Software	57	**351**
Hauppauge Computer Works	C4	**274**
Hi Tech Software	34	**376**
Integral Quality, Inc.	101	**327**
Intel Corporation	53	**190**
Kadak Products Ltd.	91	**325**
Kurtzberg Computer Systems	76	**294**
Lahey Computer Systems, Inc.	39	**186**
Language Processors, Inc.	77	**266**
Lattice, Inc.	129	**101**
Lifeboat	45	**118**
Logic Path	117	**226**
Logitech, Inc.	5	**257**
Lugaru	24	**135**
MSJ Subscriptions	100	*
M&T Catalog of Books & Software Tools	*	
Magma Systems	66	**313**
Magus Inc.	86	**336**
Manx Software Systems	7	**108**
Meridian Software Systems	33	**397**
Metagraphics	80	**392**
MetaWare Incorporated	63	**95**
Micro Way	83	**300**
MicroHelp, Inc.	19	**215**
Microprocessors Unlimited	72	**105**
Microsoft	118-119	**380**
Microsoft	120-121	**109**
Mortice Kern Systems, Inc.	90	**249**
Nanosoft Associates	82	**309**
Nantucket Corporation	135	**220**
Norton Utilities (The)	24	**243**
Oakland Group, Inc.	71	**227**
Oasys	35	**254**
Orchid Technology	15	**130**
Oregon Software	131	**357**
Periscope Co. Inc.	110	**214**
Pharlap	91	**343**
Production Language Corp.	93	**399**
Programmer's Connection	67-69	**129**
Programmers Shop (The)	40	**133**
Programmers Shop (The)	41	**301-304**
Qualstar Corporation	97	**356**
Quantum Computing	111	**144**
Quelo	84	**377**
Quilt Computing	97	**107**
Raima Corporation	103	*
68000 Toolbook	75	
SAS Institute	25	*
Scantel Systems Limited Ltd.	99	**391**
Scientific Endeavors	101	**210**
Secom Information Products Co.	49	**394**
Seidl Computer Engineering	88	**114**
Semi-Disk Systems	133	**85**
SLR Systems	93	**78**
Softfocus	79	**259**
Software Concepts Design	74	**393**
Software Garden Inc.	18	**314**
Software Security, Inc.	47	**170**
Softway	95	**372**
Solution Systems	85	**142**
Solution Systems	61	**148**
Solution Systems	61	**149**
Solution Systems	38	**152**
Texas Instruments	156	*
TRW - Command Support Division	59	*
TSF (The Software Family)	127	**230**
Tele Operating System	123	*
Turbo Tech	124	**119**
Unify Corporation	4	**332**
Vermont Creative Software	32	**157**
W.R.I.S.T. Inc.	101	**223**
Wendin	11	**112**
Workman & Associates	99	**244**
Xenosoft	84	**225**

*This advertiser prefers to be contacted directly; see ad for phone number.

Advertising Sales Offices

Southeast/Midwest
Gary George (404) 897-1923

Northern California/Northwest
Lisa Boudreau (415) 366-3600

Northeast
Cynthia Zuck (718) 499-9333

Southern California/AZ/NM/TX
Michael Wiener (415) 366-3600

Director of Marketing and Advertising
Ferris Ferdon (415) 366-3600

(continued from page 112)

plore your computer and make mighty programs. Powerful examples will lead you along the way and provide models for later reference. . . ." Such hype is commonplace from publishers' PR departments, but it's a little more unusual coming straight from the author's pen. Several paragraphs farther on, you find the admonition "The letter 'l' and the number '1' are represented in the program code by the same character. In no case is the letter 'l' used as a variable name. This should eliminate any confusion." If there's anything I'm already confused about at this point, it's why the publisher of an "ultimate" reference book would allow it out of the door containing program listings wherein the numeral *1* and the letter *l* cannot be distinguished. But no matter, let's see what the rest of the book has to offer.

Chapters 1 through 8 discuss the boot sector, file allocation table, directory structure, and file area of MS-DOS disks. Incredibly tortuous Turbo Pascal source code for a disk peeker/patcher named Explorer is developed as part of the exposition. Each chapter ends with a summary of sorts entitled "Key Programming Points." Here are the Key Programming Points listed at the end of Chapter 4: "Sectors are the smallest organizational unit. They contain 512 bytes. You can find interesting messages and modify programs by editing sectors. Explorer uses arrow keys to move the cursor."

The following few chapters discuss the partition table, "un-erasing" files, and a program that patches COMMAND.COM to change the names of MS-DOS internal commands (real useful!). Next the author covers input and output with the mouse, keyboard, and screen using IBM PC ROM BIOS drivers; an overview of file and record I/O; directory searching; and memory management. Nothing new here.

Finally, in Chapters 28 through 38, you get to the apparent reason for the book's existence: the problems and pitfalls of programming Terminate and Stay Resident (TSR) utilities. This part of the book contains much useful information, poorly organized and presented, about chaining onto interrupts, putting up and taking down pop-up displays, monitoring the keyboard for hot keys, and the like. If this part of the book were properly structured and edited, it would make a decent magazine article, but the book as a whole is a poor investment.

Norton, Peter; and Socha, John. *Peter Norton's Assembly Language Book for the IBM PC*. New York: Brady/Prentice-Hall, 1986. 413 pages with index. ISBN 0-13-661901-0.

This book is Yet Another Assembly-Language Tutorial of average quality. It is built around the design and stepwise enhancement of a simple disk sector modification utility called DSKPATCH, discussing in passing some issues of processing keyboard input and updating screen displays using the IBM PC ROM BIOS video driver.

This book is mainly notable as a demonstration of the recent trend toward commodity marketing of computers and related products. As the competition in the wonderland of silicon has become more intense, we have seen the adoption of marketing tactics that were previously the province of car, appliance, and apparel manufacturers, such as scratch-off sweepstake tickets (Borland); cash rebates (Apple); "free gifts with purchase" (with every dBASE III Plus, get a Cross writing instrument free!); mystical mumbo jumbo such as Robert Carr, Wayne Ratliff, and Jonathan Sachs being touted as "Chief Scientists" of their respective companies; and last but not least, celebrity endorsements.

Back in 1983–1984, while John Socha was a contributing editor for *Softalk/PC*, he wrote a book called *Assembly Language Safari for the IBM PC: First Explorations* (Bowie, Md.: Brady, 1984. ISBN 0-89303-321-9). For various reasons, including rather poor production values in the book itself and massive financial problems at the Brady corporate level, the book received little attention and went out of print shortly thereafter. Nowadays, John Socha works for Peter Norton Inc. in Santa Monica, California, and is the author of the program sold as the Norton Commander. When *Peter Norton's Assembly Language Book* appeared, with Peter Norton and John Socha listed as coauthors, I thought it would be instructive to make a page-by-page comparison of John Socha's old book with the new one.

A fairly generous assessment of the two books is that there are approximately 17 pages of new material in the "new" Norton/Socha version, scattered among the following topics: the *Proceed* command of DEBUG, a MASM program skeleton, MAKE files, SYMDEB, linker maps, .COM vs. .EXE programs, the *ASSUME* directive, segment overrides, and phase errors. The remainders of the two books are identical, except for some redrawn figures, the addition of some titles and divider pages, and some minor changes in wording that would have been introduced by any competent editor. There are also three new appendices that are mostly filler from other sources, such as MASM error messages and character tables. In other words, the substance of Mr. Norton's contribution to this book seems to be his picture on the cover, his billing as a coauthor, and the running head "Peter Norton's Assembly Language Book" on the top of each right-hand page.

Some might argue that the kind of misrepresentation involved here hurts no one and is therefore of no consequence. To be sure, the book's purchasers, although they might have been misled about the creative origins of the book, have still bought a reasonably useful introduction to assembly language. The three principals—John Socha, Peter Norton, and Brady—are undoubtedly happy because their ploy has caused the book to vault onto the computer best-seller lists—which means everyone involved has made some money. And last but not least, Peter Norton's recognition as an expert on all matters concerning the IBM PC has been magnified.

A Poor Man's MAKE

J. F. Philippe Marchand, of Webster, New York, sent in the programming goody of the month—a short program called CHKDATE.C. This program compares the date of two files and returns an "error level" code that can be tested within a batch file. For those people who are not fortu-

nate enough to have one of the commercial MAKE utilities (which are bundled with many of the compilers and assemblers being sold today), CHKDATE can help to automate the process of compiling and linking the various modules of an application program. Example 1, below, contains the C source code for the CHKDATE program, and Example 2, below, contains an example batch file that demonstrates the use of CHKDATE.

MS-DOS Programming Tips

George Smith, of Lilburn, Georgia, writes: "I have run across one puzzling problem [in MS-DOS 3.0]. DOS function *0eh*, which makes a specified drive the current drive, has always reported a reliable count of the number of drives on the system in register *al*. Under DOS 3.0 and later, though, the value returned in *al* is always at least 5, even if you happen to be running short-handed with fewer drives.

"I have no explanation, though I do have a solution in the form of function *DriveCnt* [Example 3, below]. The logic behind *DriveCnt* is simple. It takes the value *al* reports from function *0eh* and passes it as a drive code to function *47h* (Get Current Directory), which will set the carry flag if the drive code is invalid. It repeats calls to function *47h* with successively lower drive codes until DOS finds one it likes."

Although George has been kind enough to send in the routine *DriveCnt* for the edification of *DDJ* readers, he sells a package called Boosters for Turbo Pascal programmers that includes this subroutine and 77 others, a screen generator, some 40 example programs, and a 93-page manual. The Boosters package costs $40 and can be ordered direct from George Smith at 609 Candlewick Lane, Lilburn, GA 30247; (404) 923-6879.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 9**.

```
#include <stdio.h>
#include <sys types.h>
#include <sys stat.h>

main(argc,argv)
int argc;
char *argv[];
{ struct stat buf1, buf2;
  int result,k;
  if ( argc < 3 ) {
    printf("chkdate: usage chkdate f1 f2 .. fn\n");
    printf("     will return errorlevel 1\n");
    printf("     if f1 older than f2 .. fn\n");
    printf("     or if f1 .. fn don't exist.\n");
    exit(1);
    };
  if ( stat( argv[1], &buf1 ) != 0 )exit(1);
  for ( k=2; k<argc; k++ ) {
    if ( stat( argv[k], &buf2 ) != 0 ) exit(1);
    if ( buf1.st_atime < buf2.st_atime ) exit(1);
    };
  exit(0);
}
```

Example 1: *Phil Marchand's CHKDATE.C program*

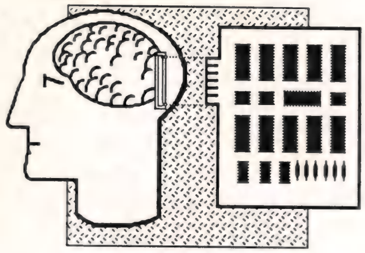
```
chkdate main.obj main.c
IF ERRORLEVEL 1 goto :compile1
goto :next1

:compile1
msc main,main;

:next1
chkdate func.obj func.c

IF ERRORLEVEL 1 goto :compile2
goto :next2

:compile2
msc func,func;

:next2
chkdate main.exe main.obj func.obj

IF ERRORLEVEL 1 goto :link
goto :exit

:link
link main.obj+func.obj,main.exe;

:exit
```

Example 2: *TEST.BAT file, a demonstration of the use of the CHKDATE program in a batch file to automate the compilation and linking of an application*

```
;         FUNCTION DriveCnt : Integer;
;         (C) 1986 George F. Smith & Company
;
;         Purpose:
;            Returns number of logical drives on host
;            computer - DOS 2.0 and above.
;
;         Sample Usage (Turbo Pascal):
;            Writeln('Number of drives is ',DriveCnt);
;
;         Suggested processing sequence:
;            MASM DriveCnt,,,
;            Link DriveCnt
;            Exe2Bin DriveCnt DriveCnt.com
;            C2I DriveCnt.com >DriveCnt.inl
;
;            (C2I utility converts .com files to Turbo Pascal
;            inline code. See DDJ, 10/86, pg. 90 )p
;
code      segment
          assume cs:code

DriveCnt:
          push  bp              ; standard
          mov   bp,sp           ; subroutine
          push  ds              ; overhead
          sub   sp,64           ; create scratch area,
          mov   si,sp           ; save in si for 47h call
          mov   ah,19h          ; get default drive
          int   21h             ; returns drive code in al
          mov   dl,al           ; move code to dl
          mov   ah,0eh          ; set default drive to itself
          int   21h             ; On return, al=# drives
;
;      If DOS 3.0 or above, number of drives in al will
;      be minimum of 5.  Will repeat calls to function 47h
;      until a valid drive code is obtained.
;
          mov   dl,al           ; number of drives in dl
          push  ss              ; segment address
          pop   ds              ;   of scratch buffer
search:   mov   ah,47h          ; is drive code okay?
          int   21h             ; carry flag set if
                                ; drive code invalid
          jnc   okay            ; jump if code valid
          dec   dl              ; drive code too big;
                                ;   decrement it and
          jmp   search          ;   try again
okay:     xor   dh,dh           ; clear dh
          mov   [bp+4],dx       ; give results to caller
          add   sp,64           ; adjust stack ptr
          pop   ds              ; restore caller's regs
          mov   sp,bp
          pop   bp
          ret
code      ends

          end   DriveCnt
```

Example 3: *George Smith's* DriveCnt *routine to determine the number of disk drives present in an MS-DOS*

Object-Oriented Programming in SCOOPS



This month, I conclude my review of PC Scheme (which I consider to be the Turbo Pascal of the PC LISP family) with some examples of object-oriented programming using SCOOPS, the object-oriented extension of the language. To do this, I'll have to make some extensions to PC Scheme itself—but first, some discussion about LISP programming in general.

LISP is the most organic and lifelike of all programming languages. Most of its dynamic character comes from the combination of complex, nested structures with dynamic reassignment of structures of pointers and a simple syntax that uses the same representation for data and programming code.

The functional programming aspect of LISP involves a special implementation of argument passing so that usually few variables need to be stored permanently. The main thing in pure functional programming is not modifying objects in permanent storage but passing symbols as if they were values being passed between mathematical functions. The main result of such a program is the structure it returns rather than the state it creates in the permanent storage of the machine.

But LISP is not simply a single-paradigm programming language. It is a language that so far has been able to absorb each new programming model as it appears and to incorporate these models in a functioning whole. Over the years it has absorbed other programming concepts and is continuing to do so—for example, as I mentioned last month, SCOOPS has assimilated some features of Smalltalk. The original model on which the LISP language was based was that of functional programming, using the lambda calculus.

by Ernest R. Tello

In pure functional LISP programming, anything that does not just return a structure but modifies the machine is generally considered a side effect. But it is often important in an object-oriented environment to modify the object hierarchy dynamically in complex and carefully controlled ways. Take, for example, the case of performing simple list processing functions, such as updating and modifying list structures. Here it is often the case that either or both of the functions of returning the necessary structure and producing the necessary structure in permanent storage are important parts of the required tasks.

In the following example, I will show various versions of a function *add-to-end* that are implemented so as to return different values and produce different side effects. This function extends the list processing functions of LISP to include the ability to add an element to the end of an already existing list structure.

The function *give-n-take* was written to demonstrate the side effects of the *add-to-end* function. First, two lists are created—*nums*, which contains the list of number words *(one two three four five)*; and *morenums*, which is composed of the complementary number word list *(six seven eight nine ten)*. Here is what *give-n-take* looks like:

```
(define nums '(one two three four
                              five))
(define morenums '(six seven eight
                              nine ten))
(define (give-n-take)
   (add-to-end (car morenums) nums)
   (set! morenums (cdr morenums)))
```

As you can see from the short session that follows, what *give-n-take* returns is different from the side effects it has on these lists. It simply returns the *morenums* list that was passed to it. When you ask LISP for the contents of these lists by typing their names at the interpreter prompt, however, you see the effects that *give-n-take* has each time it is called. It takes numbers successively from the beginning of the *morenums* list and adds them to the end of the *nums* list:

```
[2] nums
(ONE TWO THREE FOUR FIVE)
[3] morenums
(SIX SEVEN EIGHT NINE TEN)
[4] (give-n-take)
(SEVEN EIGHT NINE TEN)
[5] nums
(ONE TWO THREE FOUR FIVE SIX)
[6] morenums
(SEVEN EIGHT NINE TEN)
[7] (give-n-take)
(EIGHT NINE TEN)
[8] nums
(ONE TWO THREE FOUR FIVE SIX SEVEN)
[9] morenums
(EIGHT NINE TEN)
```

Now compare this to another version of the function called *add-to-end-2*. In the next session I create the list of integers from *1* to *4*. My original *add-to-end* returns something unusable, but the side effects are the correct result. *Add-to-end-2* does just the opposite—it returns the list with the number added to the end, but when you examine the list of integers, you see that nothing has changed.

```
[6] (define integers '(1 2 3 4))
INTEGERS
[7] (add-to-end 5 integers)
(4 5)
[8] integers
```

```
(1 2 3 4 5)
[9] (add-to-end-2 6 integers)
(1 2 3 4 5 6)
[10] integers
(1 2 3 4 5)
```

This hasn't been just an academic exercise. The side-effects version of *add-to-end* is useful for doing necessary housekeeping in an object-oriented LISP environment. It is valuable to be able to maintain lists of all the current instances of various classes that are alive in a system that is changing dynamically. Without the version of *add-to-end* that can actually modify such lists, you would not be able to update them continually.

Another function that could be useful is *delete-last!*. It performs the opposite service—that of destructively removing the final element in a list. Its definition is:

```
(define (delete-last! lst)
  (delete! (car (last-pair lst)) lst))
```

The *last-pair* function in PC Scheme returns the last pair in a list. The function works for returning the last element as a single element list because the *car* reduces the pair to a simple list structure.

The following quick session shows the behavior of *delete-last!*. As you can see, in this case both what the function returns and its side effects are identical.

```
[2] (define numbers '(one two three
                       four five))
NUMBERS
[3] (delete-last! numbers)
(ONE TWO THREE FOUR)
[4] numbers
(ONE TWO THREE FOUR)
```

Programming in SCOOPS

Sending messages is a rather simple matter of using the *send* function with the *receiver* object and the message to be sent plus its arguments. So, to send a message to the *my-body* object (introduced in the May column), giving it a new body part called *toes*, you would say:

```
[1] (send my-body put-cpart-name
                          'toes)
[2] body-parts
(HEAD NECK ARMS HANDS TRUNK LEGS
                        FEET TOES)
```

If you then changed your mind and decided to remove this new body part, you could do so by globally accessing the *body-parts* list using the newly defined function *delete-last!* as follows:

```
[3] (delete-last! body-parts)
[4] body-parts
(HEAD NECK ARMS HANDS TRUNK LEGS
                           FEET )
```

The code in Listing One, page 98, demonstrates simple inheritance in PC Scheme through several levels of a fairly linear hieararchy. First, the root class *artifact* is defined with the instance variables *material*, *weight*, *purpose*, and *cost*. Then *transport-means* is defined as a subclass of *artifact* with the additional instance variables *medium*, *time-range*, and *power-source*. Naturally, *transport-means* inherits all the variables and methods of the *artifact* class. Then *transport-vehicle* is defined as the next subclass and *passenger-vehicle* as a subclass of it. Descending further in the same linear manner of adding more and more specific classes that inherit everything from the previous class, the classes *water-transport-vehicle*, *surface-vessel*, *ship*, and *ocean-liner* are defined. The instance *object ship1* is created as an instance of the class *ship*. The two methods *speed* and *direction* are also provided for the *ship* class.

Listing Two, page 99, provides an example of multiple inheritance in SCOOPS. Here I have implemented the example used in April's column in PC Scheme. First, the classes *business* and *adversary* are defined. Then the class *competitor* is defined and uses the multiple inheritance feature to inherit everything from both of these two classes.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listings begin on page 98.)

Vote for your favorite feature/article.
Circle Reader Service **No. 10.**

The fastest C

Your search for execution speed is over. The new Microsoft® C Compiler Version 4.0 is here. With blazing performance. We've added common sub-expression elimination to our optimizer that produces code that rips through the benchmarks faster than ever before.

"...the Microsoft performance in the benchmarks for program execution is the best of the lot overall." —William Hunt, *PC Tech Journal*, January, 1986.*

But speed isn't the only edge you get with Microsoft C. Other advantages include a variety of memory models like our new HUGE model that breaks the 64K limit on single data items. Plus our NEAR, FAR and HUGE pointers, which provide you greater flexibility. All this allows you to fine tune your program to be as small and fast as possible.

"Excellent execution times, the fastest register sieve, and the best documentation in this review ...Microsoft Corporation has produced a tremendously useful compiler." —Christopher Skelly, *Computer Language*, February, 1986.

No more debugging hassles. Introducing CodeView. Free.

Now, for a limited time, we'll give you an unprecedented programming tool when you buy Microsoft C, free. New Microsoft CodeView™ offers the most powerful tool yet in the war on C bugs. Forget the hex dumps. Now you can view and work with programs at any level you want. Use the program source, the disassembled object code, or

Microsoft C Compiler Version 4.00

Microsoft C Compiler
- Produces fast executables and optimized code including elimination of common sub-expressions. NEW!
- Implements register variables.
- Small, Medium and Large Memory model libraries.
- Compact and HUGE memory model libraries. NEW!
- Can mix models with NEAR, FAR and the new HUGE pointers.
- Transport source and object code between MS-DOS® and XENIX® operating systems.
- Library routines implement most of UNIX™ System V C library.
- Start-up source code to help create ROMable code. NEW!
- Full proposed ANSI C library support (except clock). NEW!
- Large number of third party support libraries available.
- Choose from three math libraries and generate in-line 8087/80287 instructions or floating point calls:
  - floating point emulator (utilizes 8087/80287 if installed).
  - 8087/80287 coprocessor support.
  - alternate math package — extra speed without an 8087/80287.
- Link your C routines with Microsoft FORTRAN (version 3.3 or higher), Microsoft Pascal (version 3.3 or higher) or Microsoft Macro Assembler.
- Microsoft Windows support and MS-DOS 3.1 networking support.
- Supports MS-DOS pathnames and input/output redirection.

Microsoft Program Maintenance Utility. NEW!
- Rebuilds your applications after your source files have changed.
- Supports macro definitions and inference rules.

Other Utilities
- Library Manager.
- Object Code Linker.
- EXE File Compression Utility.
- EXE File Header Utility.

C Benchmarks In seconds

	Microsoft C 4.0	Lattice C 3.0	Computer Innovation C 2.3	Aztec C86 3.2	Wizard C 3.0
Sieve of Eratosthenes (register)	82.9	151.4	172.3	88.0	91.9
Copy Block	86.9	231.7	199.0	123.8	189.5

Run on an IBM PC XT with 512K memory

Microsoft CodeView
Window-oriented source-level debugger. NEW!
- Watch the values of your local and global variables and expressions as you debug.
- Set conditional breakpoints on variables, expressions or memory; trace and single step.
- Watch CPU registers and flags as you execute.
- Effectively uses up to four windows.
- Debug using your original source code, the resulting disassembly or both intermingled.
- Use drop-down menus to execute CodeView commands.
- Access the on-line help to lead you through CodeView's options and settings.
- Easily debug graphics-oriented programs since program output is kept separate from debugger output.
- Keyboard or optional mouse support.
- Enter in familiar SYMDEB or DEBUG commands.

*Reprinted from *PC Tech Journal*, January 1986, copyright 1986, Ziff-Davis Publishing.

you've ever seen.

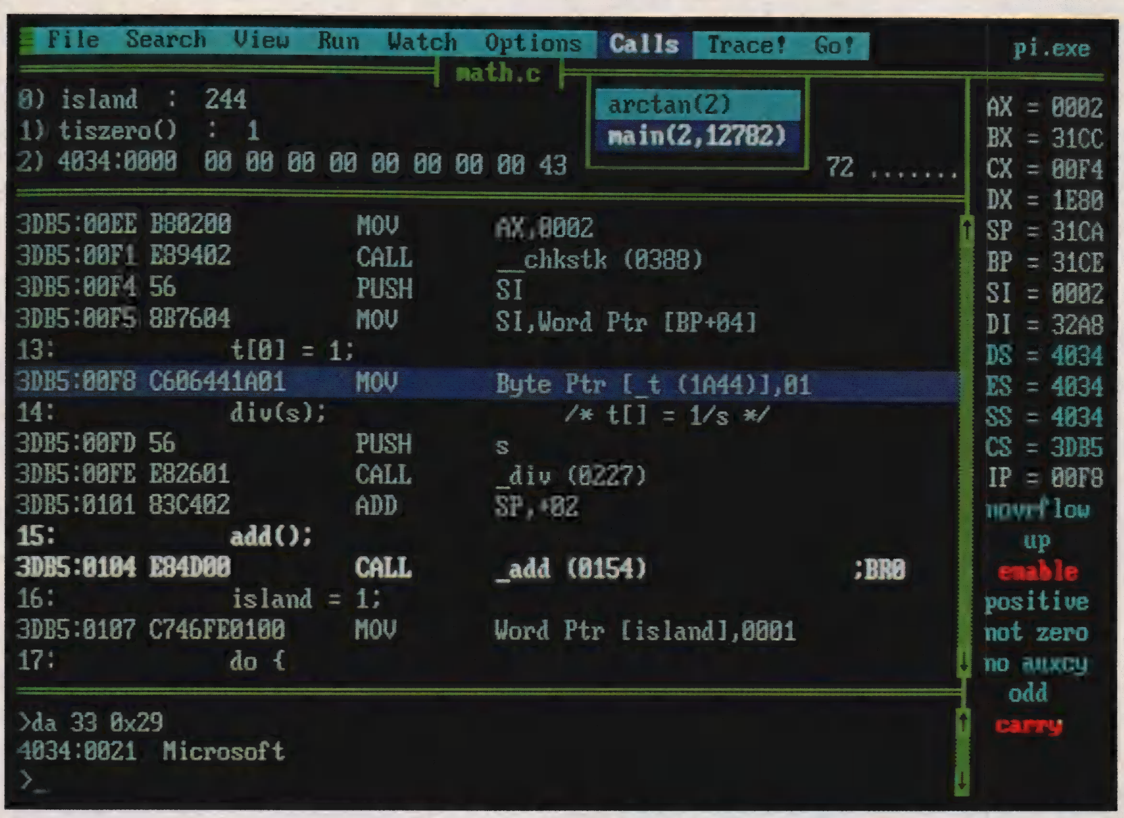
both at the same time. Open a window to view CPU registers and flags. Watch local and global variables as well. All while your program is running.

CodeView gives you complete control. Trace execution a line at a time—using source or assembly code. Or set conditional breakpoints on variables, memory or expressions. CodeView supports the familiar SYMDEB command syntax, as you'd expect. Commands are also available through drop-down menus. Combine the new window-oriented interface with our on-line help and debugging has never been easier. Or quicker.

Take the $5 CodeView tour.

You may find it hard to believe our debugger can do all we've claimed. So we're offering test drives. Five bucks will put you behind the wheel of a Microsoft C demo disk with CodeView.† See for yourself how fast debugging can get.

For more information about the CodeView demo disk, the new Microsoft C Compiler, a list of third party library support or the name of your nearest Microsoft dealer, call (800) 426-9400. In Washington State and Alaska, (206) 882-8088. In Canada call (416) 673-7638.



Microsoft® C Compiler

The High Performance Software

The fastest, tightest code.

(Though the same can hardly be said of the name.)

We have to tell you, we had a hard time getting the name down this short.

Because Microsoft's new FORTRAN Compiler actually has a far longer list of features.

It uses the same optimizer and code generator technology that made our C Compiler the industry leader.

And we've also added special loop optimizations that give you the smallest, fastest FORTRAN code a PC can handle.

"Now Microsoft's FORTRAN Optimizing Compiler generates such fast code that an IBM PC/XT approaches the speed of the VAX."

Peter Osgood, MIT, Project Athena, Director of the Real Time Lab Project.

This compiler has already passed the toughest test there is. It's been

Microsoft FORTRAN Optimizing Compiler Version 4.0.

- ♦ Uses the Microsoft C optimizing technology, plus loop optimization to generate the fastest executable code for MS-DOS. NEW!

Execution Speed (in Seconds)	Microsoft FORTRAN v. 4.0	Ryan-McFarland FORTRAN v. 2.11	IBM Professional FORTRAN v. 1.22
Sieve	7.97	9.33	38.51
Whetstone	53.82	58.67	79.04
Lookup	5.82	18.61	26.02

- ♦ Fully GSA certified for ANSI 77 compatibility with no errors at the highest level. NEW!
- ♦ Numerous IBM VS and DEC VAX extensions. NEW!
- ♦ Microsoft CodeView: Window-oriented source-level debugger. NEW!
  - —Debug using your original source code, the resulting disassembly or both intermingled.
  - —Watch and change the values of your local and COMMON variables as you debug.
  - —Set conditional breakpoints on variables, expressions or memory; trace and single step.
  - —Debug Microsoft C programs as well as Microsoft Fortran programs.
  - —Watch and change registers and flags as you execute.
  - —Easily debug graphics oriented programs since program output is kept separate from debugger output.

GSA-certified as Full ANSI FORTRAN 77, and 100% error-free.

"The Microsoft FORTRAN Optimizing Compiler let us port the 200,000 line Boeing Mathematical Library (BCSLIB) with virtually no changes. This ANSI FORTRAN 77 code was ported directly from Cray, CDC, DEC, IBM and other mainframes and workstations."

Ivor Philips, Boeing Computer Services, Program Manager Mathematical Software Libraries.

We've also included the same advanced intrinsic math functions found on VAX® and IBM® VS systems. Add improvements like our new HUGE memory model, and porting the biggest mainframe programs has never been easier.

Compiler v. 4.0 with CodeView™

Among the many additions we've made to our package is our exclusive CodeView™ windowing debugger.

It lets you trace through programs at any level you want, from source code to assembly language.

You can open windows and watch both variables (local and COMMON) and CPU registers change.

You can set conditional breakpoints using variables and expressions.

Debugging gets even easier with the compiler's advanced diagnostics. Detailed error messages are thoroughly explained and cross-referenced in our new manuals.

Documentation that has been completely revised and expanded with tons of examples.

If we're talking your language, use one of the numbers below for more details about the Microsoft® ANSI FORTRAN 77 Optimizing Compiler Version 4.0 with CodeView, and the name of your nearest dealer.

(Even if the call's toll-free, it may be a good idea to refer to it as "FORTRAN 4.0" for short.)

Microsoft® FORTRAN

The High Performance Software.

Call (800) 426-9400. In Washington State or Alaska, (206) 882-8088. In Canada, (416) 673-7638.

- Medium, Large and Huge Memory Model Libraries. NEW!
- Mix models with NEAR, FAR and new HUGE pointers.
- Common blocks and arrays greater than 64K.
- Choose from three math libraries and generate in-line 8087/80287 instructions or floating point calls:
  - floating point emulator (utilizes 8087/80287 if installed)
  - 8087/80287 coprocessor support
  - alternate math package—extra speed without an 8087/80287
- Link your FORTRAN routines with Microsoft C (v. 4.0 or higher), Microsoft Pascal (v. 3.3 or higher) or Microsoft Macro Assembler.
- Largest number of 3rd party support libraries available.
- Provides more detailed diagnostic error messages (almost twice as many as competitors) and extensive documentation with non-ANSI 77 features highlighted. NEW!
- Proven reliability—tested with over 2.5 million lines of code compiled and executed.
- MS-DOS® network support with file / record locking and sharing.
- Microsoft Program Maintenance Utility rebuilds your applications after your source files have changed. NEW!
- Other utilities including faster overlay linker (links over 1Mbyte object code), library manager, EXE file compression utility, EXE file header utility, MS-DOS environment setting utility and setup utility.

(continued from page 12)

fastest and just about the handiest editor under ten fingers (if it isn't, just redefine your macros the instant you think of something better). Just take the compiler and all-purpose resident macro program as bonuses.

Now, the importance of speed is simply that we want to work at programming, not at running an editor. Like a fine sound system, a good editor should be transparent. When I write *while v <= maxv do*, I don't think of the keys I press, only of the statement I'm producing. It should be the same when I want to move that statement, to indent it and what follows, or simply to erase the line—or to find it in a 2,000-line file. What no one wants is to sit and look at the editor editing.

Which brings us to WordStar imprinting. I got imprinted with WordStar because my mother imprinted me with ten fingers. For the first two days, it may be easier to use an editor that has F3 for "delete word" and shift-F3 for "delete line," but after those two days, only a WordStar-style editor gives you a chance to get to the point at which you only have to think "delete line" and not notice the actual fingerwork needed to do it—because it doesn't demand that your hands leave the typing position.

Hunt-and-peck chickens may not understand this, but there is a world beyond the barnyard, you know. The WordStar commands aren't meant for the user's manual but for the user's hands. They're the natural extension of sheer speed because they remove another nontransparent interface between your mind and the program text.

So, my ultimate editor is simply the Turbo Pascal editor with a good macro program plus several-document capacity, windowing, wordwrap for comments, block-limited search-and-replace with more complex specification capacity, and files greater than 64K. But never at the price of speed or the WordStar keyboard code. Don't mistake us far-voyaging mallards for clay pigeons.

Philippe Ranger
6120 Hutchison
Montreal, Canada H2V 4C2

```
ENTRY      STX   SAVEX      ;Save [X]
           ...              ;(Body of subroutine)
           LDX   #$FF       ;Restore [X]
SAVEX      EQU   *-1        ;Data field of preceding instruction
           RTS
```

Example 1: *A fast method for saving and restoring registers*

```
TXBLOCK     STX   _TXLENGTH    ;Set block length (0=256)
            STA   _TXADDR      ;Set block address
            STY   _TXADDR+1
            LDX   #-1          ;Init timeout
            LDY   #0           ;Init block index
            STY   _TXSUM       ;Reset TXSUM accum.
TXBLOOP1    LDA   $FFFF,Y      ;First, get byte to send
_TXADDR     EQU   *-2          ;Address portion of LDA
                                instruction
TXBLOOP2    BIT   HWREADY      ;Check if Tx port is ready
            BPL   TXBLOOP3     ;(N)Check timeout
            STA   HWDATA       ;(Y)Tx the byte
            EOR   #$FF         ;Accum TXSUM
_TXSUM      EQU   *-1          ;Data portion of EOR instruction
            STA   _TXSUM       ;Update running XOR sum
            INY
            CPY   #$FF         ;End block?
_TXLENGTH   EQU   *-1          ;Data portion of CPY instruction
            BNE   TXBLOOP1     ;(N)Continue sending
            LDA   _TXSUM       ;Send the checksum
            ...
TXBLOOP3    DEX                ;Timeout expired?
            BNE   TXBLOOP2     ;(N)Continue sending
            ...                ;(Y)Return timeout error
```

Example 2: *An actual 6502 code fragment*

6502 Hacks

Dear *DDJ*,

When I read Mark Ackerman's "6502 Hacks" (February 1987), I was both surprised and delighted that, in this world of 68000s and hypercubes, there is still anyone left who would spend the time and effort to write a good article about 6502 programming.

But, before anyone's programs start crashing, let me correct Mr. Ackerman on two related points. First, the software interrupt instruction (*BRK*) uses the *IRQ* (maskable interrupt) vector, not the *NMI* (nonmaskable interrupt) vector. But to make matters even worse, the *BRK* instruction pushes its address+2 on the return stack. In order to return to the opcode immediately following the *BRK*, the return address on the stack must be dug up and decremented before returning (very messy).

Second, and more important, is that the *RTI* (return from interrupt) instruction is not functionally compatible with the sequence *PLP* (pull processor), *RTS* (return from subroutine). The *RTI* instruction pulls the processor status byte and then continues execution at the address pulled from the stack, increments the address by 1, and then continues execution at the adjusted address. This compensates for the fact that the *JSR* pushes the address of the next opcode–1.

The reason for this lies in a quirk of the 6502 opcode processing. To execute the *JSR* instruction, the processor first fetches the opcode and the low byte of the address. It then pushes the current program counter (which is now pointing to the third byte of the instruction) and, finally,

fetches the high byte of the address. So, the address that gets pushed on the stack is always one byte shy of the next instruction. In contrast, the interrupt acknowledge sequence will only happen between instructions, so the progam counter is always pointing to an opcode when the interrupt return address is pushed.

In his coverage of self-modifying code, I think Mr. Ackerman missed one very useful trick. I often save registers (because they are so very scarce) by storing their contents in the data portion of a load instruction located at the end of the routine [see Example 1, page 122]. It takes 5 bytes of code but is absolutely the fastest method of saving and restoring a register (six cycles total) and both the instruction and data storage are localized in the subroutine.

To demonstrate variations of the trick, I have included a programming fragment of an actual routine [see Example 2, page 122]. The routine transmits, over an extremely fast synchronous communication port, a block of data (from 1 to 256 bytes) followed by the exclusive-ORed sum of the block. It must also make sure that the port is not "hung" by keeping a decaying timer in the *X* register. The routine is passed the address of the block in registers *A* and *Y* and the length of the block in *X*. When the loop is cooking, it can transmit a byte every 27 cycles.

Of course, I won't tell you what product this code is running in (the labels have been changed to protect the innocent), for fear that someone wouldn't buy our system if they knew I programmed this way on a regular basis.

James Bucanek
C-Si Systems
572 W. Pima
Coolidge, AZ 85228

DDJ

VIEWPOINT
(continued from page 14)

ple) to use simple trigonometry in a high-level language than it is to use it in assembly language.

High-level languages can enhance the value of a programmer's work by allowing code to run with very little alteration on many different computers—all that is required is a compatible compiler for each. My 68000 cross assembler is now operational on computers that use Z80, 8088, and 68000 processors. I would like to challenge Suman—or any assembly-language programmer—to convert an assembly-language program of similar functionality and complexity to run in two environments. I suspect that it would take them longer than the few days each that it took to port the 68000 cross assembler I presented in the April and May 1986 issues of *DDJ*.

With reference to Suman's specific criticisms, the use of named constants (for any language, including assembly language) is almost universally considered to improve maintainability by localizing changes. Although I used the constants *FIRST* and *LAST* only twice (not once, as Suman stated), I feel their use was justified.

As to the lack of initialized variables in Modula-2, I agree that this is a defect of the language. That should not, however, be an indictment of all high-level languages—C does allow initialized variables, including initialized structures and arrrays. If Modula-2 allowed initialized variables, there would have been no need for *InitOperationCodes* (the module that Suman disliked so much). The mnemonic lookup table could have been created at compile time—in the module that needed the table—and with much less overhead.

Certainly, the mnemonic lookup table of X68000 could have been written more efficiently using assembly language. It would have been harder to write or to read, however, and it would not have been portable (some processors invert the order of bytes within words). The sets that were used for the allowable addressing modes (*ModeA* and *ModeB*) represented the 68000 addressing scheme in a recognizable way. For example, one of the set members was called *Size67* to indicate that the sixth and seventh bits of the 68000 operation code were used to indicate the size of the operation. If that member were to be converted to a computer word (1s and 0s), a programmer reading the listing might be at a loss to figure out what the significance of a particular bit (or combination of bits) was.

Finally, Suman seems to contradict his own point about turgidity and redundancy when he suggests that I should have used 118 individual *write* statements instead of the simple loop:

```
FOR i := FIRST TO LAST DO
  WriteRec (f, Table68K[i]);
END;
```

Perhaps an ideal solution to the dilemma of choosing between the expressiveness of high-level languages and the efficiency of assembly language is to mix the two. Most decent compilers allow the integration of assembly-language modules. A program can be written and debugged in Modula-2 or C, then profiled to identify the bottlenecks. Finally, these sections can be rewritten in assembly language. The original high-level language code can be left behind as a comment block to the assembly language. Although this would hamper portability somewhat, the sections of a program that benefit from recasting in assembly language are usually a small portion of the overall code. Projects that would be Herculean tasks in assembly language become quite comfortable using modern high-level languages. Get your project working, then if you need more speed or better memory utilization, tune it up using a small amount of hand-coded assembly language.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 1**.

DR. DOBB'S C TOOLBOX

Dr. Dobb's Toolbook of C

This authoritative reference contains over 700 pages, including the best C articles from Dr. Dobb's Journal along with new material. You'll find hundreds of pages of valuable C source code, including a complete compiler, an assembler, and text processing programs.

Toolbook of C — Item #005 — $29.95

Small C Handbook & Small C Compiler

This compiler and handbook provides all you need for learning how compilers are constructed, and for learning C at its most fundamental level. You'll find a discussion of assembly language concepts and program translation tools, and of how to generate a new version of the compiler. Full source code is included. Please specify MS/PC-DOS or CP/M: Kaypro, Osborne, Apple, Zenith Z-100 DS/DD, 8" SS/SD.

CP/M Compiler & Handbook — Item #006B — $37.90
MS/PC-DOS Compiler & Handbook — Item #006C — $42.90

Small Windows: A Windowing Library for Small C

Small-Windows is a complete windowing library for Small-C. The package contains: 18 video functions written in assembly language; 7 menu functions that support both static and pop-up menus; and 41 window functions, including functions to clean, frame, move, hide, show, scroll, push, and pop windows. A file directory facility illustrates the use of the window menu functions and provides file selection, renaming and deletion capability. Two test programs are also included. For PC/MS-DOS systems only. Documentation and full source code is included.

Small-Windows — Item #109 — $29.95

Small-Tools: Programs for Text Processing

This package of Small-C programs performs specific, modular operations on text files. It is supplied as source code with full documentation. With the Small-C Compiler, you can select and adapt these tools to meet your needs. Please specify MS/PC-DOS or CP/M: Kaypro, Osborne, Apple, Zenith Z-100 DS/DD, 8" SS/SD.

Small-Tools — Item #010A — $29.95

Small-Mac: An Assembler for Small-C

This package includes: a simplified macro facility, C language expression operators, object file visibility, descriptive error messages and an externally defined instruction table. Source code and documentation is included. CP/M only. Please specify: Kaypro, Osborne, Apple, Zenith Z-100 DS/DD, 8" SS/SD.

Small-Mac — Item #012A — $29.95

Special Packages

CP/M **C** Package
Save Over $27!

Receive: Dr. Dobb's Toolbook of C, The Small-C Handbook and Small-C Compiler, Small-Mac Assembler, and Small-Tools Text Processing Programs. Only $99.95!

CP/M Package — Item #005A — $99.95

MS/PC-DOS **C** Package
Save $22!

Receive: Dr. Dobb's Toolbook of C, The Small-C Handbook and MS/PC-DOS Addendum, Small-C Compiler, Small-Tools Text Processing Programs and Small Windows. Only $109.95!

MS/PC-DOS Package — Item #005W — $109.95

C Disk Formats
Please indicate MS/PC-DOS or CP/M. For CP/M specify: Apple, Osborne, Kaypro, Zenith Z-100 DS/DD, 8'' SS/SD.

To Order: Return this order form with your payment to **M & T Books, 501 Galveston Dr., Redwood City, CA 94063.**
Or, call TOLL-FREE 800-533-4372 Mon–Fri 8AM-5PM.
(In CA call 800-356-2002)

ORDER FORM

Name ______________________

Address ______________________

City ______________ State ________ Zip __

Item #	Description	Price

Subtotal ________

CA residents add sales tax ____ % ________

Add $2.25 per item for shipping ________

TOTAL ________

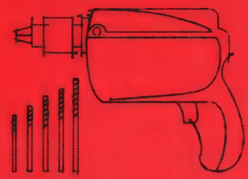
☐ Check Enclosed. **Make Payable to M & T Publishing Inc.**

Charge my ☐ VISA ☐ M/C ☐ Amer. Exp.

Card # ______________ Exp.Date ________

Signature ______________________

3128C



POWER TOOLS for SYSTEM BUILDERS™

TSF is owned and operated by programmers, so we understand your needs. We believe and practice integrity in all business dealings. We advertise real prices and offer the best possible terms to all customers. We provide prompt delivery of current product versions.

We carry only products which we have personally used or which come with strong recommendations from developers we respect. We have the technical capability and the interest to search for and find new products to meet your specific needs. We will gladly provide quotes for volume purchases or for products outside our normal product line. We consider it our job to help you get answers when publisher's technical support is unresponsive. At TSF service means more than a pushy sales pitch and a $2 lower price.

We accept checks, Visa, Mastercard, American Express and COD. We charge your card only when we ship and do not add a surcharge. We provide free UPS delivery on software orders over $100 ($3 delivery charge on orders under $100). Our COD fee is $4. We allow return privileges on most products. Give us a try.

June Special: BDT only $75 -- Save 25%

Basic Development Tools™

Powerful "Automatic Programming" Tools that Save You Hours of Valuable Time.

The novice or power programmer can easily add these professional features.

- Screen Builder
- B+ Tree Disk File Access
- EZ Screen Pop-Up Windows
- Help screen message system

BASIC Development Tools™ (BDT™) is a set of four compatible tools which greatly increase the quality of your programs while requiring a minimum investment in money, code and debugging time.

1. Screen Builder System

Translates a painted screen image into BASIC code that you merge or include into your programs.

2. B+Tree Data Manager

Provides a very fast file access system that can be used for both sequential access and for random access with up to 40 keys. Complete source provided.

3. EZ Screen Pop-up Window Manager

Lets you easily insert and remove menus, windows and notepads from your program's display. Assembler routines (which you call from Basic) are used to automatically save and restore the display buffer, providing full screen management without slowing down execution speed.

4. Help Message System

Allows the creation of context sensitive help messages in your application program.

BDT gives you four great tools to improve the quality of your programs and to improve your own programming productivity. The BDT tools can be used used separately or together and are compatible with BASICA, GWBASIC, Quick Basic™ and Borland Turbo Basic™. Includes two diskettes and a 220 page manual. No royalties. At a list price of $99 its a bargain -- at TSF's June special price of $75 its a steal. Don't delay, call or write now to place your order! 30 Day money back guarantee.

STERLING CASTLE™

Sterling Castle, 702 Washington St., Suite 174, Marina del Rey, CA 90292

The Software Family

649 Mission Street
San Francisco, CA 94105

(1)800-443-7176

In California or outside U.S.

(415)583-4166

Send Or Call For Comprehensive Free Catalog

TSF carries a complete line of software and hardware tools for programmers and system engineers. We offer "big name" products at competitive prices and a large selection of hard-to-find products that save you hours of research, evaluation and development work. Our catalog includes hype-less product descriptions that explain what our products do to improve your productivity and your product's quality. Call or send the coupon for your free copy.

__ Please send me a free catalog.
I'm most interested in:
__ Basic __ C __ Pascal
__ Turbo Pascal __ Other Languages
__ dBase __ Technical Publishing

__ Please send the following products:

____

Name: ______________________

Address: ______________________

City/State/Zip: ______________________

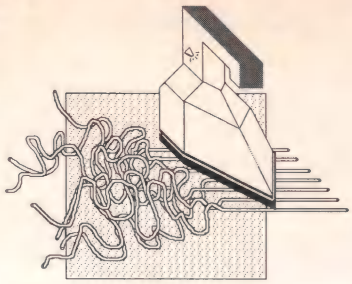
Credit Card: ______________________

Expiration Date: ______________________

Mail to: TSF, 649 Mission Street, San Francisco, CA 94105

CIRCLE 230 ON READER SERVICE CARD

THE STATE OF BASIC



BASIC Functions

In this issue we'll look at user-defined functions as implemented by QuickBASIC, Turbo BASIC, True BASIC, and Better BASIC.

QuickBASIC 2.0 permits user-defined nonrecursive functions to extend over multiple lines. All function names must start with the letters *FN*. A data type symbol may be required, depending on the data type returned by the function and any global default name declarations used. A function has an optional list of scalar arguments (no arrays are allowed) with all the arguments passed by value. Multiline functions end with an *END DEF* statement and can use *EXIT DEF* to exit from the function.

Most of the new BASIC dialects allow passing of all arguments by value. A function with no arguments that returns a simple value is one way of implementing Pascal-like constants; you cannot accidentally alter the function's value, as may be the case with a variable.

Nonparameter variables that are used within the functions are global. To avoid undesirable side effects and to localize such variables, include them in a *STATIC* declaration. This technique ensures that new addresses are assigned to these variables every time a *STATIC* declaration is encountered.

Turbo BASIC functions are similar to those of QuickBASIC, with the following differences:

- Turbo BASIC functions can be recursive.
- Local variables are declared inside functions using the *LOCAL* keyword.
- The *SHARED* keyword is used to explicitly declare global variables.

Turbo BASIC offers *LOCAL* and *STATIC* declarations, which give programmers the ability to explicitly declare local nonstatic variables. In Turbo BASIC, local arrays are first declared in the *LOCAL* list and then dynamically dimensioned using *DIM DYNAMIC*.

True BASIC implements user-defined functions in a slightly different way from that of the previous two BASIC dialects. First, function names do not have to start with the letters *FN*. The price to pay, however, is the use of *DECLARE DEF* declarations to inform True BASIC of the function

```
DEF FNFACT(N%)
' QuickBASIC nonrecursive factorial function
STATIC I%, F ' static local variables
F = 1 ' Initialize
' loop to get factorial
FOR I% = 2 TO N%
 F = F * I%
NEXT I%
FNFACT = F
END DEF
```

Example 1: *QuickBASIC listing for a nonrecursive factorial function*

```
DEF FNFACT(N%)
' Turbo BASIC recursive factorial function
IF N% > 1 THEN
  FNFACT = N% * FNFACT(N%-1)
ELSE
  FNFACT = 1
END IF
END DEF
```

Example 2: *Turbo BASIC listing for a recursive factorial function*

```
DEF Fact(N)
! True BASIC recursive factorial function
IF N > 1 THEN
  LET Fact = N * Fact(N-1)
ELSE
  LET Fact = 1
END IF
END DEF
```

Example 3: *True BASIC listing for a recursive factorial function*

```
REAL FUNCTION: Fact
INTEGER ARG: N
EXTERNAL: Fact
10 IF N > 1 THEN RESULT = N * Fact(N-1) ELSE Fact = 1
END FUNCTION
```

Example 4: *Better BASIC listing for a recursive factorial function*

```
REAL FUNCTION: LOGN
REAL ARG: X
REAL ARG: BAISE/OPT=10
10 RESULT = LOG(X) / LOG(BAISE)
END FUNCTION
```

Example 5: *Simple power function in Better BASIC that demonstrates the default parameter feature*

names imported from external libraries and modules. True BASIC supports recursive multiline functions that take scalar- and array-type parameters. All the function parameters are passed by value.

In essence, True BASIC supports two levels of functions: internal and external. Internal functions are those located in the main program, before the unique *END* statement. All the variables in the internal function and not in the argument list are global, which enables internal functions to create and manipulate global variables. External functions are located either after the *END* statement of the main program or in an external library or module file. External functions defined in modules can access information through the argument list, *SHARED* variables, and *PUBLIC* variables. External functions in libraries have a strict data interface because they rely mainly on the argument lists. Argument lists in True BASIC can contain a file I/O channel number that allows file I/O, which provides another method of accessing large data that are stored in intermediate files.

Better BASIC approaches the implementation of functions in a Pascal-like fashion: the type of the function or its arguments is explicitly declared using data-type keywords and not symbols. Much of the discussion about Better BASIC procedures that appeared in the May column applies to functions. In addition:

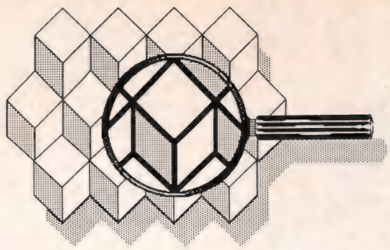
- Function names do not need to start with the letters *FN*.
- The function type and name are declared on a separate line. This causes Better BASIC to respond interactively by creating a new workspace for the function and display and by displaying the memory available for it. Consequently, functions can use any range of valid line numbers without conflicting with other functions, procedures, and the main program.
- Parameters are declared by first listing their type, the keyword *ARG:*, and the parameter name. Like Better BASIC procedures, parameters can be assigned default values.
- The result of the function is returned using the standard identifier *RESULT*.
- Recursive functions need to declare the function and any local variables as external. This enables Better BASIC to allocate new addresses instead of using the same ones as in the calling function.

Examples 1–4, page 128, show versions of the factorial function written in each of the BASIC implementations discussed here. Recursive versions are used with all implementations except QuickBASIC. Example 5, page 128, shows a simple Better BASIC function that returns the logarithm to any base. The default is base 10. That is, using the function *LOGN(X)* returns the base-10 logarithm, and *LOGN(X,BAISE)* returns the logarithm of *X* to base *BAISE*.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 12**

OF INTEREST



Languages

Computer Crossware Labs has introduced Real BASIC, a BASIC interpreter for the Atari 520 and 1040 ST that executes BASIC code 20 to 100 times faster than ST BASIC while maintaining full compatibility. It contains an in-line Motorola-compatible assembler that allows you to switch to assembly language without leaving the interpreted BASIC environment and to have full access to BASIC variables while in assembly mode. Real BASIC's features include an integrated, full-screen editor that supports ST BASIC commands and some Micro Emacs commands; extended graphics instructions; and function calls that deal with the mouse and joystick ports. Real BASIC sells for $69.95. Reader Service No. 16.
Computer Crossware Labs Inc.
516 Fifth Ave., Ste. 507
New York, NY 10036
(212) 677-3686

Digitalk has released two optional extension kits to accompany Release 1.2 of Smalltalk/V for PC-DOS machines. The first kit integrates full EGA color capabilities into the Smalltalk/V environment, greatly enhancing the system's bit-mapped graphics. The second kit, called Goodies, offers several new programming kernels and lets you extend Smalltalk to handle applications that require discrete event simulation, forward-chaining interface operations, and connection to external sensors and instrumentation. Release 1.2 of Smalltalk/V is priced at $99, and updates to 1.1 can be downloaded from CompuServe and BIX or obtained on disk from the company for $10. The optional extension kits cost $49 each. Reader Service No. 17.
Digitalk Inc.
5200 W. Century Blvd.
Los Angeles, CA 90045
(213) 645-1082

Rational Visions is shipping a full-featured PROLOG programming system for the Atari ST. The system uses the Edinburgh standard syntax, making it compatible with most popular tutorials on PROLOG. Features include an Emacs-style text editor within the system's interpreter that allows you to interact with external disk files, the internal knowledge base, or invoke PROLOG goals from within the editor; a built-in grammar-rule translator that allows the development of natural-language interfaces; support for floating-point values; and extended math functions. Also, all PROLOG's metaprogramming primitives have been implemented, as have Atari-specific features such as GEMDOS primitives and interfaces to VDI and AES. The system is not copy-protected and supports user-written applications free of licensing restrictions. The package sells for $39.95. Reader Service No. 18.
Rational Visions
7111 W. Indian School Rd., Ste. 131
Phoenix, AZ 85031
(602) 846-0371

Datalight has introduced Optimum-C, a full-featured global optimizing C compiler for IBM PCs and compatibles that supports the Unix System V C language along with several proposed ANSI extensions. Features include 8087 and software floating-point support, strong type checking, ROMable code generation, a MAKE program, MS-DOS object files format, Lattice C compatibility, and full library/start-up source code. Support is provided for compact-, small-, medium-, and large-memory models. Source licenses and support contracts are available for the compiler. Optimum-C sells for $139. Reader Service No. 19.
Datalight
P.O. Box 82441
Kenmore, WA 98028
(206) 367-1803

Tools

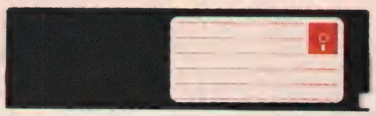
Devpac Amiga, from **HiSoft**, is a 68000 program development system for the Amiga computer that features a combined editor and assembler and a symbolic disassembler/debugger. The full-screen editor can be used from either the mouse or from the keyboard. The assembler is a complete, fast, macro assembler that supports include files read from disk, conditional assembly, and Motorola-standard macro handling and that can produce executable or linkable code. The debugger includes all the expected commands, such as breakpoints and single-stepping, and also allows you to use your original symbols when debugging programs. It uses its own screen for its display so as not to disturb that of other programs. Devpac Amiga is available in the United States from Apex Resources. It runs on any Amiga with at least 512K RAM and costs $99.95. Reader Service No. 20.
Apex Resources
129 Sherman St.
Cambridge, MA 02140
(800) 343-7535
In MA (617) 876-2505

Numerical Algorithms Group (NAG) now offers 172 selected routines from its extensive NAG Fortran Library for use on workstations and personal computers. The NAG Fortran Workstation Library provides the routines most frequently used by engineers and PC programmers. Each routine includes an example program, the source, and data results to illustrate the routine's usage. The NAG Fortran Workstation Library is available for the DEC MicroVAX, IBM PC line, and Sun workstations. License fees range from $1,296 for a single workstation to $384 each for 11 or more. Reader Service No. 21.
Numerical Algorithms Group Inc.
1101 31st St., Ste. 100
Downers Grove, IL 60515
(312) 971-2337

P-tral, from **Woodchuck Industries**, is a native code BASIC-to-Pascal translation package that translates IBM BASICA/MS BASIC to Turbo Pascal. The program is interactive and lets you pick out or name subroutines as well as rename variables not fitting Pascal criteria. The program works

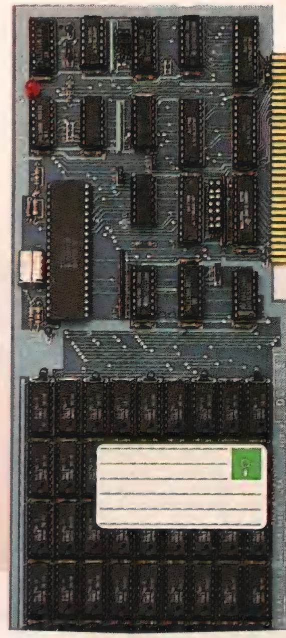
Think fast! Pick the better fit...

Our 5th Year Bonus!
Mention this ad when ordering
and get your choice of a V 20-8
(replaces 8088) or $20 off on
your Battery Backup purchase!



FLOPPY DISK.

- Fills time between coffee breaks
- Makes a hard disk seem *fast*
- Your computer appears busy (even if you aren't!)
- Wears out moving parts



SEMIDISK Disk Emulator.

- Gets that job done NOW
- Makes a hard disk seem *slow*
- Maximizes your productivity with anything from databases to compilers
- Totally silent operation

...for YOUR demanding tasks.

SURPRISE! *Neither* is memory mapped, so they don't affect your precious Main Memory. *Both* retain data indefinitely - even with the computer turned off.

THE SEMIDISK SOLUTION. You could invest in a series of "upgrades" that turn out to be expensive band-aids without solving your real problem. Even those "Accelerator" and "Turbo" boards do little to speed up disk-bound computers. If your applications spend too much time reading and writing to disk (and whose don't?), you won't want to settle for anything less than a SemiDisk disk emulator. The SemiDisk comes in 512K and 2Mb capacity. More boards may be added to make up to an 8 Megabyte SemiDrive!

SPEED THAT'S COMPATIBLE. PC, XT or AT, if you need speed, the SemiDisk has it. How fast? Recent benchmarks show the SemiDisk is from 2 to 5 times faster than hard disks, and from 25% faster (writing) to several times faster (random reads) than VDISK and other RAMdisk software that gobble up your main memory.

MEMORY THAT'S STORAGE. Using our small external power supply, with battery backup, your data remains intact through your longest vacation or even a seven-hour power failure!

CELEBRATE WITH US! Now, SemiDisk celebrates its fifth birthday with a special offer for IBM-PC owners. Buy a SemiDisk now and we'll include an 8 MHz V-20 microprocessor (replaces the 8088) to make your new SemiDisk run even faster. Don't need the V-20? We'll take $20 off the price of your Battery Backup Unit!

	512K	2Mbyte
IBM, PC, XT, AT	**$495**	**$ 795**
Epson QX-10	**$495**	**$ 995**
S-100 SemiDisk II	**$795**	**$1295**
S-100 SemiDisk I	**$299**	-------------
TRS-80 II, 12, 16	**$495**	**$ 995**
Battery Backup	**$130**	**$ 130**

**Someday you'll get a SemiDisk.
Until then, you'll just have to...wait.**

SemiDisk

MasterCard VISA

SemiDisk Systems, Inc., 11080 S.W. Allen Blvd., Beaverton, Oregon 97005 (503) 626-3104

CIRCLE 85 ON READER SERVICE CARD

(continued from page 132)

best with a hard disk and requires MS-DOS or PC-DOS 2.1 or later with ANSI.SYS. P-tral sells for $179. Reader Service No. 22.
Woodchuck Industries Inc.
340 W. 17th St., #2B
New York, NY 10011
(212) 924-0576

RTC Plus, a FORTRAN- and RATFOR-to-C translator package from **Cobalt Blue**, allows you to tap old FORTRAN code for new C development. The translator is designed for translating non-I/O FORTRAN libraries and code in which I/O is concentrated in a few routines. Also, more than 95 percent of STUG's RATFOR statements are supported by RTC Plus for complete translations of clean RATFOR code. RTC Plus runs under MS-DOS, Version 2.2 or later and costs $325. Reader Service No. 23.
Colbalt Blue
1683 Milroy, Ste. 101
San Jose, CA 95124
(408) 723-0474

TurboPower Software has released an upgrade to T-DebugPLUS, a symbolic run-time debugger for Turbo Pascal. The new version (1.04) allows Turbo Pascal programmers to debug code in overlays and access CPU registers and memory. Other improvements include easier customization of the debugger and a new MAKELST utility program that creates a commented disassembly listing of Turbo Pascal programs. T-DebugPLUS requires Turbo Pascal 3.0 and a PC-DOS machine with 256K RAM. It sells for $60 ($10 for upgrade). Reader Service No. 24.
TurboPower Software
3109 Scotts Valley Dr., Ste. 122
Scotts Valley, CA 95066
(408) 438-8608

Laney Systems has released StruBAS 2.0, a structured BASIC toolkit for application development with QuickBASIC and IBM BASIC 2.0 compilers in network and single-user environments. Integrated tools include enhanced structured programming facilities, screen handling, a Btrieve interface, StruBAS/ISAM for those not wishing to buy Btrieve, an object library of utility subroutines, and utility programs. A preprocessor expands BASIC with new commands to produce a more readable and structured language while making all BASIC's features and constructs available as well. StruBAS programs are translated by the preprocessor to compilable BASIC, which is then compiled and linked to produce executable programs. StruBAS is not copy-protected, and no royalties are charged for programs written using the package. It requires a PC-DOS machine with 320K RAM and a hard disk and sells for $495. Reader Service No. 25.
Laney Systems Inc.
3 Office Park Dr., Ste. 105
Little Rock, AR 72211
(501) 225-7755

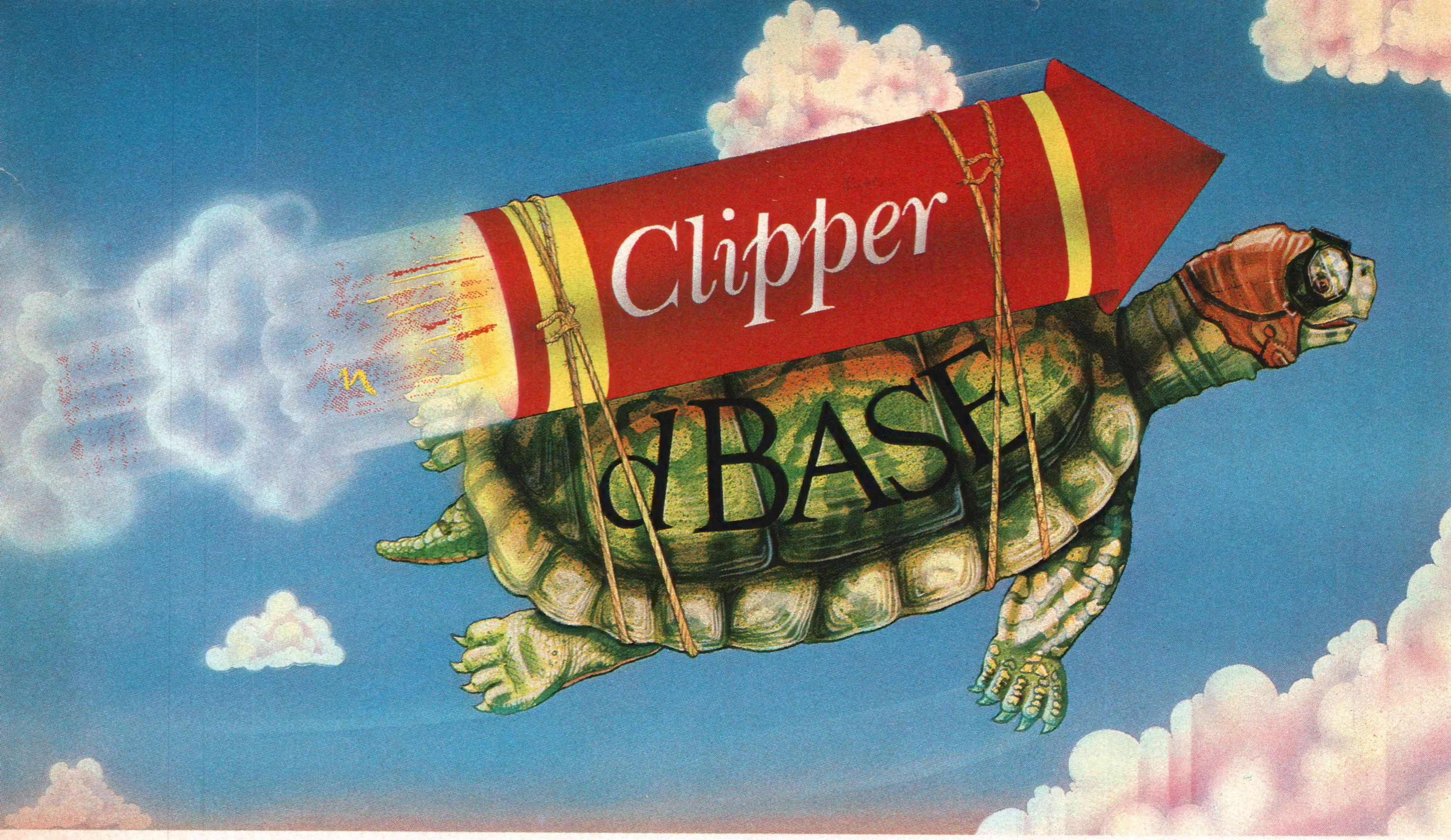
BASIC Program Analyzer, from the Expert Systems Division of **Expert Information Systems**, is a rule-based system that reads Microsoft's GW-BASIC program files in ASCII format and performs analysis, listing, and cross-reference functions upon them. The analyzer prints individually tabulated cross-references of line numbers, variables, keywords, quoted strings, and numerical constants found in theograms. Error, warning, and comment messages are inserted where applicable into tabulated cross-references, and an ASCII file is produced of all portions of the printout, including the cross-references. Optional parameters can be used to modify some of the analyzer's operations, such as testing for compiler syntax compatibility and for redirecting output. BASIC Program Analyzer is available for most MS-DOS computers and costs $99.95. Reader Service No. 26.
Expert Information Systems Inc.
Expert Systems Division
P.O. Box 1310
El Campo, TX 77437-1310
(409) 543-9222

Libraries

United States Software Corp. has released a 68HC11 floating-point library. This complete math package operates on the 68HC11, is designed in a modular format, and is delivered in source assembly language for maximum flexibility. The library conforms to the IEEE 754 Floating Point Standard. The 68HC11 FPAC/DPAC is available in either single-precision or double-precision formats. The library includes trigonometric, logarithmic, and exponentiation functions, and data-conversion and floating-point utility procedures. It is available on IBM PC-compatible media for a one-time fee of $950. Reader Service No. 27.
United States Software Corp.
14215 N.W. Science Park Dr.
Portland, OR 97229
(503) 641-8446

LINLIB, from **Information and Graphic Systems**, is a library of C routines for developers of scientific and statistical software. The package performs all basic operations on vectors and matrices, with no limitation on size of vector or matrix. Of major importance are the routines that factor matrices (LU, QR, and Cholseky factors). These factorizations enable you to solve any kind of system equations—underdetermined, determined, and overdetermined. Also included in LINLIB are routines based on the B-spline for manipulating splines. LINLIB sells for $150. Reader Service No. 28.
Information and Graphic Systems
15 Normandy Ct.
Atlanta, GA 30324
(404) 231-9582

DDJ



Turtle Souped

Real programmers don't use dBASE. Or do they?

We're finding that some very swift programmers are using it to write some very fast applications, and are completing their projects much more quickly.

But they cheat.

They use our Clipper™ compiler to combine dBASE™ with C and assembler.

With dBASE used like pseudo-code, they can then quickly create prototypes that actually run.

Then, with dBASE doing the high-level database functions, they use our Clipper compiler to link in C or assembly language modules from their own bag of tricks.

And they're finding that they're linking in less than they expected because Clipper compiled code runs so fast and because of Clipper's built-in enhancements.

Clipper includes easy networking that provides file and record locking the way it should be done.

Fast screens that can be treated as memory variables and eliminate the need for direct screen writes and all that tortuous heap management code.

Box commands that make windowing a breeze. And more.

So if you'd like to use your time more productively, check us out: Nantucket Corporation, 12555 W. Jefferson Boulevard, Los Angeles, CA 90066.

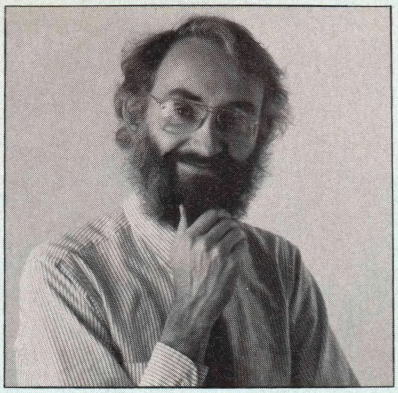
Or if you're on deadline, call (213) 390-7923 today.

Clipper could get you out of the soup.

Nantucket®

CIRCLE 220 ON READER SERVICE CARD

SWAINE'S FLAMES



I think I may be the first person who has ever actually seen a computer bug.

It came to me at the end of a long and disillusioning night when my body, no doubt in sympathy with the new punitive tariffs on Japanese goods, began rejecting a dinner of overripe sushi. En route from bathroom to loft, I heard an odd tapping sound from the office and stopped to investigate.

In my collywobbled state I had neglected to turn off the Macintosh, and what I saw in the glow from the screen was a bug—it looked a lot like a cockroach—repeatedly climbing to the top of the Mac and leaping to the keyboard to enter a character. The exertion must have been tremendous, and he looked pretty beat when he finally finished and crawled into the disk drive. I'm not surprised he didn't bother with capitalization or punctuation (the bottom row of keys was almost too long a jump for him).

He left this behind.

boss i m afraid that
the computer curmudgeon john
dvorak has been talking
about our beloved dr dobb s journal
of whatever again
if dr. dobb s became
outspoken it would be
dangerous dvorak said in one of
those tasty magazines you
leave lying around this office i
wish you would leave a
crust of bread i m getting tired
of paper and glue

i wonder does he mean
like that 1940s science fiction
writer was dangerous
when he explained how
an a bomb would work or like the
white house press corps would be
dangerous if it decided it
wasn t the white house s press corps
or like sesame street would be
dangerous if it taught children to
question their teachers
and their parents
or maybe like infoworld was
dangerous when it was
publishing dvorak

sometimes i think that john
dvorak is a modeless
dialog box the max
headroom of the
personal computer industry

how could dr dobb s become
dangerous is what i
d like to know
maybe by examining legislator s
voting records on software
issues or by doing a study of
compiler pricing vs value or by
publishing detailed specs on
data formats of
commercial products so that data
held hostage by obsolete
applications could be liberated
or by showing how to develop
software for the 386
without waiting for you
know who or by questioning
the need for an operating system
on a personal computer at all

i used your modem to
call dvorak s number and talked
with his dog
sparky maybe dr dobb s should
list the companies
that preannounce products to
outflank their competitors
sparky suggested
or do their beta testing in
release version one point oh
or cripple their products with
copy protection tricks
or refuse to eliminate
known bugs what s this about
bugs i asked him but
he just barked

one thing i know is that dr
dobb s could stand to
be a little more rough and
tumble a bit more capricious and
corybantic there s a
dance in the old dame yet
and if that s dangerous it s
fine by me please save
this file as i can t work
the mouse nor would i want to i
haven t always had the best of
relations with mice yours archy

Scarab journalism was invented by Don Marquis, a *New York Sun* columnist, when he found a cockroach composing in his office late one night some seventy years ago, learned that the bug was a reincarnated vers libre poet, and cannily conned the poet roach into meeting copy deadlines for him for the next ten years.

The saga of archie the cockroach itself allegedly owes its existence to the newspaper account of a typewriting rat who plied his or her art by night circa 1916 in a Dobb's Ferry, New York, garage. By some odd symmetry in the transmigration of souls, a Dobb has once again played host to verse vermin.

Fine by me.

Michael Swaine

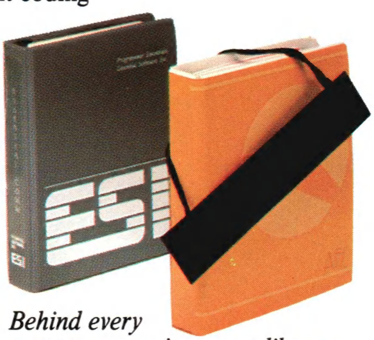
Michael Swaine
editor-in-chief

You'd Be Surprised Who Uses Essential Graphics Functions

And some of our well-known customers would be just as surprised if they saw themselves in this ad. We don't splash their names all over the place as a matter of professional courtesy.

Let's face it. Just because someone else uses a product is not reason enough to buy it. The clincher is that our programs run a *documented 40% faster* than the closest competitor. To complete the picture, our code is up to *75% smaller* due to efficient coding and the granularity of functions.

However, should curiosity get the best of you, call us at 201-762-6965 and we'll drop a few highly impressive names on you.



Behind every great program is a great library.

Functions At A Glance

Features:
- Fastest functions available
- Dots, Lines, Circles, Arcs, Pies, Bars
- Manipulate character fonts
- Move blocks, do animation
- User definable patterns
- Seed filling in a boundary
- Clipping on screen coordinates

Devices Supported:
- IBM, Epson, Oki printers
- HP Plotters, HP Laser Jet
- Microsoft, Logitech Mice

Graphics Adapters:
- IBM Color Graphics
- IBM Enhanced Graphics
- Hercules Graphics
- AT&T, Olivetti Graphics
- Tecmar Graphics Master
- Others (Call)

Compiler Compatibility:
- Microsoft, C, Fortran, Pascal
- Lattice C, Aztec C
- Computer Innovations C86, DeSmet C
- Wizard C, Mark Williams

$250.00

Draw Your Own Conclusions

When you're responsible for a project that includes advanced graphics, "graphics windowing," or character font manipulation, Essential Graphics is the clear choice. We've taken the grind out of graphics programming and replaced it with speed and versatility.

No Royalties, 30 Day Guarantee

We believe that selling you a programming tool does not make us your co-authors. So we don't charge any royalties or run time fees. If within 30 days you don't find our library satisfactory, dump the whole thing and receive a complete refund.

Do Your Homework

The library you buy can influence the rest of your programming life. We encourage you to do some checking before making a decision. When you've done your homework, you'll choose Essential. Call our support staff of experienced C programmers and find out before you buy how things will be after your check clears.

To order or for support call: 201-762-6965

For foreign orders contact:

England: Gray Matter Tel. (0364) 53499
Japan: Lifeboat Inc. of Japan Tel: 293 4711
West Germany: Omnitex Tel. 07623-61820

Essential Software, Inc.

P.O. Box 1003, Maplewood, New Jersey 07040

CIRCLE 138 ON READER SERVICE CARD

HAUPPAUGE

Thrust your math into high gear!

HAUPPAUGE has a hunch you need faster math calculations. So we've launched our 287 FaST10—the highest speed math coprocessor available for PC/AT's and all AT compatibles.

Now you can speed up your math calculations without worrying about data safety. Our FaST10 has a self contained high speed clock that speeds up math coprocessor operations without jeopardizing accuracy.

The FaST10 accelerates programs including 1-2-3, Framework, AutoCAD and Symphony (to name just a few)—by as much as 250%!

Our FaST10 is compatible with: 6 or 8 MHz IBM PC/AT's, the Compaq DeskPRO 386 and all 286-based PC's like the Compaq DeskPRO 286 & Portable 286.

We have all the Coprocessors you need!

FaST modules, which operate at full speed, for the PC/AT, Compaq DeskPRO 386 and PC/AT compatibles:

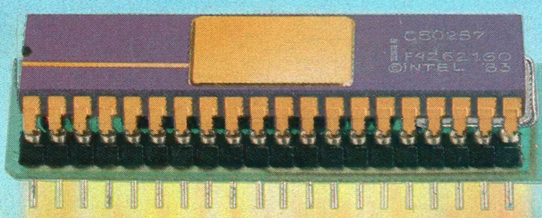
287 FaST 10—full speed 10MHz **$469**
287 FaST 8—full speed 8 MHz **$379**
287 FaST 6—full speed 6 MHz **$249**

Math coprocessors for AT compatibles and accelerator cards:

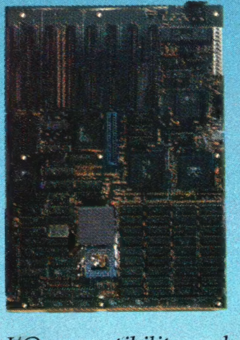
287-10 Chip **$439**
287-8 Chip **$349**
287-6 Chip **$219**

Math coprocessors for PC/XT's & higher speed compatibles:

87 Chip—4.77 MHz ... **$129**
87-2 Chip—8 MHz **$195**



Save your PC from early retirement. Get Hauppauge's new 386 MotherBoard! Hauppauge has a smart way to give your IBM PC or XT the speed and compatibilty of the 80386.



It's our 386 MotherBoard. With a 32-bit, 16 MHz 80386 processor and 1 Mbyte of high speed memory, it scores high on desktop publishing, CAD graphics, engineering and business applications. 386 MotherBoard is a motherboard replacement that's 100% form fitting. So installation takes just 20 minutes.

To accept new generations of DOS, we gave it PC/AT BIOS, I/O compatibility and up to 16 Mbytes of high speed expansion RAM.

And our 386 MotherBoard has a lot more on the ball than simple accelerator cards. Because in addition to speed, it has a 16 bit expansion slot. Plus a 32 bit slot for high speed memory expansion. There's also a 387 math coprocessor socket for the ultimate in high speed math calculations, and a clock/calendar with battery backup for true PC/AT compatibility.

386 MotherBoard also comes with very intelligent prices:

386 MotherBoard/PC **$1495**
386 MotherBoard/XT **$1495**
387 Math coprocessor **$695**

Hauppauge

Your math coprocessor specialists

Hauppauge Computer Works, Inc.
358 Veterans Memorial Highway,
Commack, N.Y. 11725
Phone: 516-360-3827

Trademarks: QuickBASIC: Microsoft, Calculus: AI Ware, HFT+: TechWare, MathTOOLBOX: Syntel, Labtech Notebook: Laboratory Technology Corp., PC and PC/AT: IBM Corp.

CIRCLE 274 ON READER SERVICE CARD